DEEP LEARNING FOR SCIENTIFIC DATA REPRESENTATION AND GENERATION

A Dissertation of Dissertation

Submitted to the Graduate School
of the University of Notre Dame
in Partial Fulfillment of the Requirements
for the Degree of

Doctor of Philosophy

by

Jun Han

Chaoli Wang , Director

Graduate Program in Computer Science and Engineering
Notre Dame, Indiana
March 2022

DEEP LEARNING FOR SCIENTIFIC DATA REPRESENTATION AND GENERATION

Abstract

by

Jun Han

Scientific visualization (SciVis) is one of the core components in supporting fundamental scientific discoveries and engineering designs. For example, scientists perform numerical simulations and produce 3D scalar and vector data to visualize, analyze, and understand various kinds of natural phenomena, such as climate change and star formation. However, the cost of these simulations is expensive when time, ensemble, and multivariate are involved and the scientific data are presented in diverse forms including streamline, pathline, stream surface, volume, and isosurface. A core problem in SciVis is how to efficiently and effectively produce and analyze these diversified data. In this dissertation, I develop novel deep learning methods to enable more effective and efficient frameworks for scientific data representation and generation.

In scientific data representation, I propose a unified framework that processes both streamlines and stream surfaces through auto-encoder decoder structure. Moreover, I build an interface that allows users to explore the relationships between the learned features and visual representations. I also utilize geometric deep learning (e.g., graph neural network) to extract node-level and surface-level features in an unsupervised fashion for node clustering and surface selection tasks. In scientific data generation, I introduce a comprehensive pipeline for variable selection and translation through feature learning, translation graph construction, and variable translation. This framework can serve as a data extrapolation

and compression solution to reduce simulation costs. Besides, I develop an end-to-end generative framework that can synthesize spatiotemporal super-resolution volumes with high fidelity. Further, to improve network generalization, I propose an unsupervised pre-training stage using cycle loss. This spatiotemporal super-resolution approach can upscale data up to 512 times in spatial dimension and 11 times in temporal dimension, which offers scientists an option to reduce data storage.

CONTENTS

FIGURES

TABLES

ACKNOWLEDGMENTS

This dissertation would never be completed without the support from my advisor, my collaborators, my friends, and my family.

First of all, I would like to express my sincere gratitude to my advisor Dr. Chaoli Wang, who helped me build the background in scientific visualization and encouraged me to explore different directions and target for high-impact works. Dr. Wang always provided insightful guidances and suggestions on every project and spent tremendous effort to revise my papers. He also taught me how to manage time during a project and how to express ideas effectively.

I also want to thank Dr. Dany Chen, Dr. Walter Scheirer, and Dr. Hanqi Guo for serving as my committee members and providing many helpful feedbacks to improve this dissertation.

I am also thankful to Dr. Jun Tao, who worked closely with me on my first project and helped me establish the foundation in building graphical user interface and visualization system. And it is my honor to work with my colleagues and friends, Hao Zheng, Hongxiao Wang, Yang Zhang, and Zhichun Guo. I learned a lot from them and the conversations with them can always inspire me to discover new directions and ideas.

I would also like to thank Dr. Danny Chen, Dr. Hanqi Guo, Dr. Jun Tao, Dr. Kwan-liu Ma, Dr. Wei Chen, and Dr. Xiaoru Yuan for their help and advice when I was seeking an academic job position.

I am also grateful to Li Li, Liyao Liu, Shuan Yang, Xiangtao Wang, and Jinglan Gu for their guidances and helps in my life and study. They are extremely charming, knowledgeable, and kind. I always get inspired and refreshed after talking to them. They are role

models for me. It is my.great honor to meet with them and I will always be proud to be a part of the Effective Oral English (EOE) family.

I am also be lucky to be supported by many great friends. Ming He and Yue Hu are my close friends for more than ten years, who keep pulling me out from my stressful Ph.D. life. We have a lot of wonderful and joyous moments. I can still remember the days when we celebrate our birthdays together and share our happiness and pleasure. Chao Qin and Yi Zhang are my dear friends that I can keep talking and expressing my feelings and thoughts with, especially when I felt unhappy and discouraged.

I would also like to thank Wenlong Zhang, Fei Jiang, Zhengya Zhang, Yini Zhang, Junhao Wei, Yuhao Zhang, and Hai Zhu for their spiritual supports.

Finally, I would like to thank my parents Jianxin Han and Yan Zhang for supporting me through all these years.

The research in this dissertation was supported in part by the National Science Foundation through grants IIS-1455886, CNS-1629914, DUE-1833129, IIS-1955395, IIS-2101696, and OAC-2104158.

CHAPTER 1

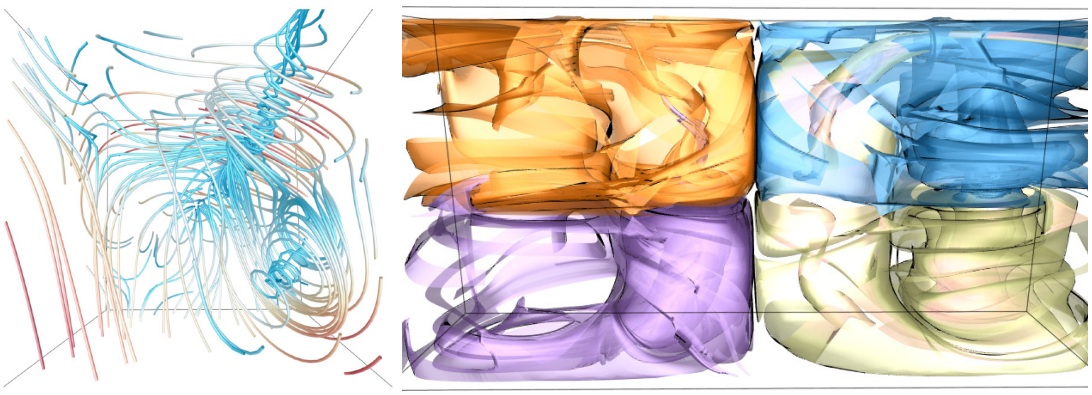
INTRODUCTION

Scientists perform numerical simulations and produce 3D scalar and vector data to visualize, analyze, and understand various kinds of natural phenomena, such as climate change and chemical reactions. A typical scientific visualization pipeline contains two stages: simulation and post-hoc analysis. In simulation, domain scientists solve partial differential equations on supercomputers given a set of initial conditions (e.g., pressure, Reynolds number, and Kinematic viscosity) to produce ensemble, multivariate, and time-varying volumetric data. After that, scientists design algorithms and interfaces to analyze and visualize the data. In particular, large-scale simulations often produce thousands of ensembles, thousands of time steps, and tens of variables with high resolution. Generating, storing, analyzing, and visualizing a huge amount of data poses three main challenges.

During post-hoc analysis, the same data (e.g., vector data) may require diverse representations for visualization and analysis. For example, streamlines and stream surfaces are two visual forms to represent the underlying flow. However, existing flow analysis approaches focus on either streamlines [143, 170] or stream surfaces [29, 132], there is no unified solution. Moreover, a single stream surface can encompass two-level information. For instance, it can include both global information (such as the overall shape) and local information (such as individual sources or sinks). No approach has been designed to extract local information for better visualizing and understanding steam surfaces. I focus on two different aspects to address the mentioned problems. First, I use a single convolutional neural network to learn hidden representations for both streamlines and stream surfaces and these features can help us identify the representative set of lines and surfaces. Second,

I formulate the surface as a mesh structure and design a graph convolutional network to learn the two-level information.

Running simulations is time-consuming and expensive and the limited I/O bandwidths cannot match the rate of data production. As such, scientists could only afford to sparsely store the outputs at the spatial, temporal, and variable levels. Techniques for generating scientific data accurately allow scientists to obtain the data quickly without waiting for days. In addition, these techniques reduce the storage cost while preserving the data evolution and the relationships among different variables and ensembles. While previous approaches have introduced various solutions for data generation, e.g., bicubic interpolation, histogram matching, most of them can not tackle complex volumetric data and the generated results may not preserve spatial (e.g., structure and texture) and temporal (i.e., close similarity among neighboring time steps) coherence. I focus on deep learning-based data generation approaches to enable scientists to recover high-fidelity data from different aspects (i.e., spatial, temporal, and variable).



(a) streamline rendering (b) stream surface rendering

Figure 1.1: Example of streamline rendering (a) and stream surface rendering (b) using the five critical points and Bénard data sets, respectively.

1.1 Steady Vector Data

One kind of scientific data is vector data. Each voxel stores the velocity components (i.e., u, v, w). There are two common approaches to visualize and understand a vector data: streamline rendering and stream surface rendering. A streamline is a curve tangent to the flow field everywhere and a stream surface is a continuous surface that is everywhere tangent to the vector it passes, which can be obtained from streamlines traced from a densely seeded curve [1]. An example of these two renderings are shown in Figure 1.1.

1.1.1 Unified Deep Learning Framework for Streeamline and Stream Surface Clustering and Selection

Many challenges exist when it comes to generating representative flow lines or surfaces as well as visually exploring a large collection of flow lines or surfaces. Although seeding and selection of streamlines have been well studied, the same problem for stream surfaces is clearly underexplored. All existing approaches for effective line and surface seeding and selection *explicitly* make use of handcrafted features (e.g., entropy, curvature, torsion, saliency, critical points, separation lines, vortex cores) in their solutions. I instead, take a drastically different approach that automatically learns features from the input lines or surfaces and encodes them *implicitly* in a latent space. This is achieved by borrowing techniques from deep learning that has made a significant impact on many fields including those that are closely related to scientific visualization, such as computer vision and computer graphics.

I aim to automatically extract latent features from raw streamline or stream surface data, which can be achieved using an auto-encoder. Note that generative adversarial network is capable of generating novel data from a given data set that look at least superficially authentic to human observers, which does not directly match our purpose. I choose the sparse and stacked version of the autoencoder framework.

In Chapter 3, I introduce FlowNet, a single deep learning approach for streamline and stream surface clustering, filtering, and selection [51]. The key lies in the design of an autoencoder that automatically learns line or surface feature descriptors. I show that by carefully designing the network architecture and loss function, the features learned can be used to well reconstruct the lines or surfaces with minimum errors. To visually explore the features, I perform dimensionality reduction and apply different clustering algorithms. I further develop a visual interface along with intuitive and convenient interactions to enable users to effectively explore the underlying set of streamlines and stream surfaces.

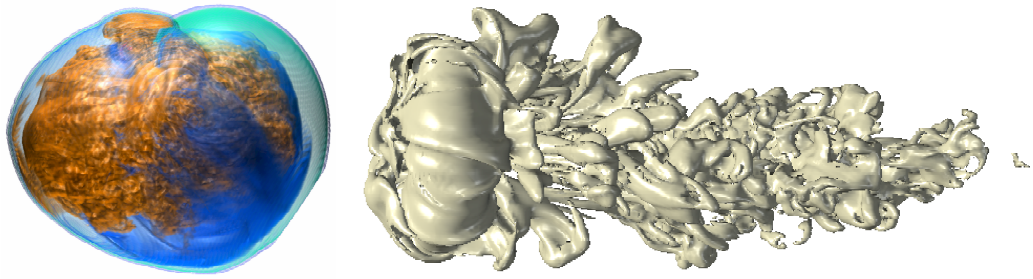
1.1.2 Geometric Deep Learning for Node and Surface-Level Representation Learning

Achieving node clustering and surface selection needs to answer three key questions. First, *how to represent surfaces effectively*? Although multiview-based [119, 139] and voxel-based representations [51, 165] have been proposed to formulate a surface, these methods have their disadvantages. Voxel-based representation is space-wasting and computationally expensive since most voxels carry rather localized information when it comes to encoding surface features. Multiview-based representation is impractical for stream surfaces since flow patterns could be severely self-occluded. Furthermore, this representation can only extract *surface-level* information (such as the overall shape) rather than *node-level* information (such as individual sources or sinks). Second, *how to group nodes on a surface without any given labels*? Unlike 3D objects, where many node labels are supported, it is impractical to manually label every node on the surface since this process is extremely time-consuming and expensive. Third, *how to derive surface features from a set of node features*? Although several existing approaches [56, 175] can directly generate surface features, they require labeling each graph, which is not suitable in our scenario as we need to handle a large set of surfaces (e.g., $1,000$).

To respond, I apply a geometric representation [15] (i.e., meshes) for surfaces since it can preserve geometric information of the nodes. We introduce SurfNet, a deep learning

approach for embedding nodes on surfaces. These learned node embeddings support node clustering, surface clustering, filtering, and selection. The crux of SurfNet is the design of a *graph convolutional network* (GCN) and a novel loss that can automatically learn the hidden feature of every node on a single surface in an *unsupervised* fashion. Specifically, we train SurfNet to learn an end-to-end function that maps a node feature with only its information (e.g., position, normal) to a node feature with its neighborhood information. The trained model allows us to explore the relationship among different nodes on a single surface or the relationship among different surfaces.

In Chapter 4, I leverage geometric deep learning framework to embed node-level and surface-level information by designing an unsupervised geometric loss [48]. These learned embeddings can be utilized in node clustering and surface selection tasks.



(a) direct volume rendering (b) isosurface rendering

Figure 1.2: Example of direct volume rendering (a) and isosurface rendering (b) using the supernova and argon bubble data sets, respectively.

1.2 Multivariate Time-Varying Volumetric Data

Multivariate time-varying volumetric data (MTVD) demonstrate how multiple variable scalar fields evolve over time. They usually have tens of variables and each variable has

thousands of time steps. Researchers often apply volume visualization techniques to visualize a single variable at a specific time step. In general, there are two common approaches to render volumetric data: direct volume rendering and isosurface rendering. The former maps voxels to color and opacity through a set of predefined parameters (i.e., transfer function and viewpoint). The later extracts geometric features (i.e., triangles) from the volume given an isovalue. Figure 1.2 shows an example of both rendering results.

1.2.1 Variable Selection and Translation

Translating one variable sequence to another variable sequence poses four main challenges. First, understanding the relationships among different variables in MTVD is critical for variable-to-variable (V2V) translation. Choosing two arbitrary variables for translation could lead to unexpected results since the randomly chosen variables may exhibit dramatically different patterns. Therefore, *variable selection* should be considered so that high-quality V2V translation can be achieved. Second, once the transferable variables are determined, choosing the appropriate source and target variables is still crucial since the translation difficulty could vary given a different variable as input. Third, unlike volume temporal and spatial super-resolution tasks that aim to interpolate data through their neighborhood information, V2V translation performs *extrapolation* instead of *interpolation*, which is a much more difficult task. Fourth, both *global* and *local* information must be considered simultaneously as multivariate temporal patterns in different regions are non-linear and non-uniform. Assuming the translation is local and linear may not ensure acceptable results: producing blurred features and resulting in fewer details in the visualization (i.e., direct volume rendering and isosurface rendering).

To tackle these challenges, I propose a novel solution for addressing the V2V translation problem for MTVD analysis and visualization, inspired by image-to-image translation tasks and representation learning techniques. V2V is a comprehensive framework for selecting transferable variables and synthesizing variable sequences. I leverage GANs to

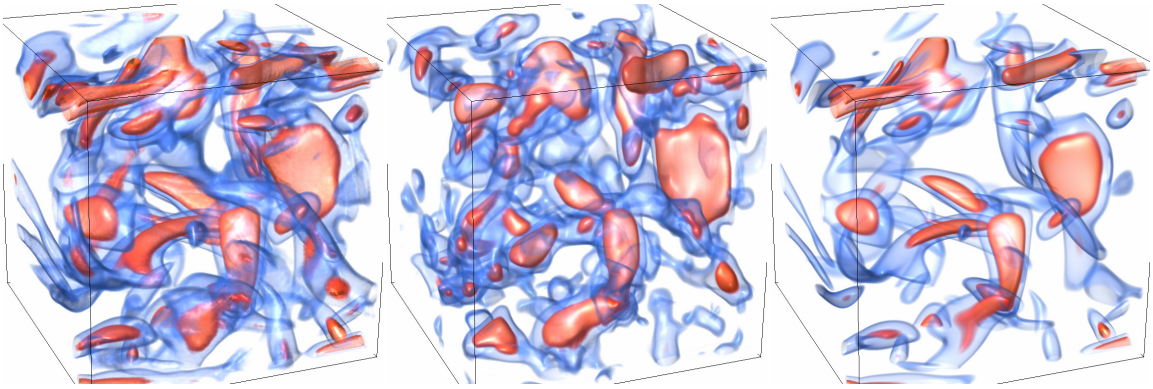
learn the variable mapping non-linearly and non-locally. Our solution consists of three stages: *feature learning* (aiming to find the relationships among different variables for MTVD), *translation graph construction* (aiming to detect the source and target variables), and *variable translation* (aiming to learn a mapping function from one variable to another variable). The training data could be obtained at earlier time steps from the two variable sequences. During inference, V2V can synthesize a variable sequence conditioned on another variable at later time steps.

In Chapter 5, I present a three-stage solution (i.e., feature learning, translation graph construction, and variable selection) to recover one variable sequence conditioned on another variable sequence [52]. We also propose a deep learning solution for translating scalar field into vector fields [42].

1.2.2 Spatiotemporal Super-Resolution

Three challenges remain for the spatiotemporal super-resolution (STSR) task. First, although SSR and TSR have been studied independently, merely concatenating the two solutions cannot guarantee satisfactory STSR volumes since the rendering quality is far away from GT. An example is shown in Figure 1.3. Designing an *end-to-end* STSR architecture is critical for avoiding error accumulation and amplification. Second, deep learning's success depends heavily on large input data, which is often challenging to acquire in scientific visualization. The limited training data will prevent the network from a better generalization during inference. How to utilize inadequate data samples to improve the generalization power should be considered in the optimization. Third, the computational cost is high as it usually requires days to train a GAN on 3D data. In addition, the synthesized volumes should maintain similar spatial (e.g., structure and texture) and temporal (e.g., close similarity among neighboring time steps) coherence compared with GTs.

To respond, I design **STNet**, an *end-to-end* **s**patio**t**emporal generative **net**work for STSR. STNet encompasses two stages: *pre-training* and *fine-tuning*. During pre-training,



(a) SSR+TSR (b) TSR+SSR (c) GT

Figure 1.3. STSR using two solutions: SSR+TSR and TSR+SSR.

STNet accepts all available low-resolution data as input, generates synthesized low-resolution volumes, and applies cycle loss for optimization. During fine-tuning, STNet takes low-resolution data at early time steps as input, produces STSR volumes, and leverages volumetric and adversarial losses for training. Specifically, I first investigate popular framework designs in both SSR and TSR tasks and design an end-to-end spatiotemporal model with post-upsampling for spatial upscaling and feature interpolation for temporal upscaling. That is, STNet interpolates the feature of each low-resolution volume and upscales the features into the data (super-resolution) space. Second, I customize a *pre-training* task for STSR by only leveraging the information from low-resolution volumes. The goal is to explicitly promote a better generalization for producing spatiotemporal volumes for time-varying data. Third, I design a spatiotemporal discriminator to guarantee spatial and temporal coherence of the synthesized spatiotemporal volumes. I apply a two-stage optimization procedure to cut the computational cost and boost the stability of GAN training.

In Chapter 6, I will propose an end-to-end spatiotemporal super-resolution solution combined with pre-training techniques to further improve the generalization of deep learning models [53]. I also studied the spatial and temporal super-resolutions for scalar [45, 46] and vector [43, 47] fields.

Besides the mentioned works, I investigated the problem of reconstructing steady [50] and unsteady [41] from streamline using deep learning, and volume completion [49].

1.3 Organization

This dissertation is organized as follows. Chapter 2 introduces related works and discusses how our approaches relate to and differ from others. Chapter 3 establishes FlowNet, a unified deep learning-based framework that allows users to interactively explore streamlines and stream surfaces in 2D space for clustering and selection. Chapter 4 presents a graph convolutional network for learning node and surface embeddings from surface in an unsupervised fashion. Chapter 5 introduces V2V, a deep learning solution for variable selection and translation, which are based on computing variable similarities in the feature space and translating variables using GAN. Chapter 6 describes an end-to-end approach for synthesizing spatiotemporal super-resolution volumes and a pre-training algorithm to improve network generalization. Chapter 7 discusses several future directions, including lightweight model design, physics-informed deep learning, disentangled learning, federated learning, etc.

CHAPTER 2

RELATED WORK

In this chapter, I provide an overview of previous works on machine learning for scientific visualization (including flow visualization, volume visualization, and multivariate relationships), data recovery and representation with deep learning techniques, pre-training techniques, and geometric deep learning.

2.1 Machine Learning for Scientific Visualization

There is a growing body of works that apply deep learning to solve scientific visualization problems. Tzeng et al. [149, 150] pioneered the use of artificial neural networks for classifying 3D volumetric data sets. Ma [98] pointed out the use of neural networks as a promising direction for visualization research. With the explosive growth of modern deep learning techniques, researchers have recently started to explore the capabilities of *deep neural network* (DNN) to address various problems.

2.1.1 Flow Visualization

Hong et al. [63] leveraged *long short-term memory* (LSTM) to estimate the access pattern for parallel particle tracing. Wiewel et al. [164] proposed an LSTM-based solution to predict the changes of pressure fields for learning the temporal evolution of fluid flows. Kim and Günther [76] extracted steady reference frames from unsteady 2D vector fields through a two-step CNN. Jakob et al. [70] designed a CNN for interpolating flow maps by creating a large 2D fluid flow field as training samples. Guo et al. [43] designed CNNs that

generate super-resolution of 3D vector field data in the spatial domain with a scaling factor of 4 or 8. Werhahn et al. [163] established two GANs for fluid flow super-resolution: one upscales slices, which are parallel to the *xy*-plane, and the other refines the whole volume along the *z*-axis working on slices in the *yz*-plane. Eckert et al. [28] presented ScalarFlow, a deep learning solution for accurate physics-based reconstructions from a small number of videos. Prantl et al. [118] proposed two-stage deformation-aware neural nets that learn the weighting and synthesis of dense volumetric deformation fields for the space-time representation of physical surfaces from liquid simulations. Kim et al. [77] presented a GAN model to synthesize fluid simulations from a set of reduced parameters and designed a novel loss function that guarantees divergence-free velocity fields.

2.1.2 Flow Line and Surface Selection

In flow visualization, selecting representative flow lines has become a useful alternative of seed placement. For *view-dependent* streamline selection and filtering, Marchesin et al. [99] measured the contribution of each streamline to the understanding of the vector field and selected those streamlines that have a higher contribution and lower probability leading to visual clutter. Ma et al. [97] presented an importance-driven approach that ensures coherent streamline update when the view changes gradually. For *view-independent* streamline clustering and selection, Yu et al. [176] clustered streamlines hierarchically and formed streamline bundles as representatives that succinctly capture flow features and patterns at varying levels of detail. Tao et al. [143] selected streamlines by considering their contributions to all sample viewpoints. Both streamline selection and viewpoint selection can be achieved using a unified information-theoretic framework which builds two interrelated information channels between a pool of streamlines and a set of sample viewpoints.

Lu et al. [96] advocated a distribution-based approach and utilized dynamic time warping to define the similarity between streamlines for clustering and query. Oeltze et al. [111] evaluated three different kinds of clustering techniques (k-means clustering, agglomerative

hierarchical clustering, and spectral clustering) in terms of clutter reduction when visualizing streamlines traced from simulated blood flow.

Only a few works address the issue of flow surface selection. Martinez Esturo et al. [32] favored stream surfaces where the flow is aligned with principal curvature directions. Simulated annealing is used to select a globally optimal stream surface based on a set of stream surface quality measures. Schulze et al. [132] extended the above work to select a set of globally optimal stream surfaces in an iterative manner. All selected surfaces are mutually distant to convey different flow features while reducing visual occlusion and clutter.

2.1.3 Volume Visualization

Zhou et al. [186] presented a convolutional neural network-based (CNN) solution for volume upscaling which better preserves structural details and volume quality than linear upscaling. Raji et al. [122] leveraged CNNs to iteratively refine a transfer function, aiming to match the visual features in the rendered image of a similar volume data set with the one in the target image. Cheng et al. [20] presented a deep-learning-assisted solution which depicts and explores complex structures that are difficult to capture using conventional approaches. Berger et al. [12] designed a generative neural network (GAN) to compute a model from a large collection of volume-rendering images conditioned on viewpoints and transfer functions. Shi and Tao [136] proposed a CNN-based viewpoint estimation method that achieves good performance on images rendered with different transfer functions and rendering parameters. Xie et al. [167] designed a temporally coherent approach to generate spatial super-resolution volumes where temporal coherence is guaranteed through a temporal discriminator. Weiss et al. [162] presented an image-space solution that learns to upscale a sampled representation of geometric properties of an isosurface at low resolution to a higher resolution. They considered temporal variations by adding a frame-to-frame motion loss to achieve improved temporal coherence. He et al. [60] presented InsituNet, a

rendering image-based generative network that takes ensemble and rendering parameters as input and produces the corresponding volume rendering images as output. Tkachev et al. [147] established a CNN for detecting irregular behaviors in spatiotemporal volumes, which assists in selecting time steps and analyzing ensemble dissimilarity.

2.1.4 Multivariate Relationships

Researchers have studied point-wise correlation coefficients [17, 37, 120, 140] and gradient similarity measure [131]. Wang et al. [153] studied the information flow between variable pairs using transfer entropy for investigating variable causal relationships. Biswas et al. [13] classified variables using *surprise* and *predictability* derived from information theory and leveraged a graph-based representation for variable exploration. Liu et al. [94] designed the probabilistic association graph based on the *informativeness* and *uniqueness* concepts to uncover the hidden associations between different variables. Tao et al. [144] considered isosurface similarities across the time and variable dimensions for time-varying multivariate data and designed the matrix of isosurface similarity map for visual exploration.

2.2 Data Representation and Generation

2.2.1 Representation Learning

Representation learning is a focused goal of many deep learning solutions. For example, Girdhar et al. [36] utilized an autoencoder to learn representative features of 3D objects for producing novel 3D objects and the corresponding 2D images. Chen et al. [19] designed LassoNet that attempts to learn a latent mapping from viewpoint and lasso to point cloud regions for lasso selection of 3D point clouds. Porter et al. [117] established a CNN to select representative time steps for time-varying multivariate data. Hao et al. [184] built a representative slice selection approach based on image features extracted by deep learn-

ing models for improving the image segmentation quality. Zhang et al. [182] proposed a feature-based metric for evaluating the generated images across different architectures and tasks and the metric can better capture human perceptual similarity judgement.

In Chapter 3, I propose a unified framework for representative streamline and stream surface selection using encoder and decoder structure. Beside the deep learning solution, a novel interface is also designed for user to explore the relationship between learned representation and object.

2.2.2 Temporal and Spatial Super-Resolution

Deep learning has achieved great success in generating temporal super-resolution for video. For instance, Niklaus et al. [109] introduced a CNN for frame interpolation where the network learns a kernel through the input frames and then applies the learned kernel to generate the missing frames. Nguyen et al. [108] proposed a deep linear embedding model to interpolate the intermediate frames. They transformed each frame into a feature space, then linearly interpolated the intermediate frames in the feature space, and finally recovered the interpolated features to the corresponding frames. Jiang et al. [72] established a CNN to estimate the forward and backward optical flows via two given frames. They then wrapped these optical flows into the frames to generate arbitrary in-between frames. Liu et al. [95] proposed Deep Voxel Flow, a deep learning-based solution for interpolating video frames by estimating optical flows without supervision. Meyer et al. [102] conducted PhaseNet, a CNN-based solution, that can directly estimate the large motions and phase decomposition of the intermediate frames.

In spatial super-resolution, for images, Dong et al. [27] proposed an autoencoder that upscales single images with an upscaling factor of three (i.e., the scaled image is nine times of the original one in size). Ledig et al. [83] established a GAN with residual blocks to infer photo-realistic natural images for an upscaling factor of four. Zhang et al. [183] proposed a deep CNN with a channel attention mechanism to achieve better visual single

image super-resolution results. For videos, Sajjadi et al. [130] utilized the early synthesized high-resolution frames to predict the subsequent frames through a recurrent video super-resolution solution. This treatment yields temporally-consistent results while reducing the computational cost. Jo et al. [73] proposed a DNN that generates dynamic upsampling filters and a residual image so that the low-resolution image can utilize the dynamic filter to generate a high-resolution image directly, and the computed residual can refine the structural details. Pérez-Pellitero et al. [115] established an adversarial recurrent network for video upscaling, where a temporally-consistent loss is computed through estimating the optical flow of two frames to guarantee the temporal coherence among different frames.

In Chapter 6, I propose an end-to-end generative framework for producing high-quality spatiotemporal volumes with a cycle loss-based pre-training algorithm.

2.2.3 Paired Image-to-Image Translation

Deep learning solutions have achieved great success in image-to-image translation tasks, such as image colorization and style transfer. Zhang et al. [180] designed a deep learning solution that produces vibrant and realistic colorful images conditioned on gray-scale images. Zhang et al. [181] established a CNN which directly maps a gray-scale image, along with sparse, local user "hints" to an output colorization and propagates user edits. Isola et al. [69] utilized conditional GANs for studying various image-to-image translation problems, such as aerial-to-map, day-to-night, and edge-to-photo. Park et al. [112] proposed a spatially-adaptive normalization layer for synthesizing photo-realistic images based on a semantic layout. Wang et al. [156] proposed few-shot vid2vid framework which requires few example images during training through a novel network weight generation module utilizing an attention mechanism.

In Chapter 5, I show how to apply representation learning and GAN to select transferable variables, detect transferable variable order, and translate one variable volume sequence to another variable volume sequence.

2.3 Network Pre-Training

Pre-training in deep learning models aims to provide a good parameter initialization for better generalization in a particular task (e.g., classification). Based on different tasks, pre-training examples include inpainting [114], colorization [82], synthesis [26], feature agreement [40], rotation prediction [35], context prediction [24], and feature contrast [18, 59]. Unlike these pre-training approaches, which are tailored for classification, detection, or segmentation, in Chapter 6, I propose a novel unsupervised pre-training method for STSR using cycle loss [177, 187].

2.4 Geometric Deep Learning

Geometric deep learning can be mainly classified into two categories: spectral and spectral-free solutions [15].

2.4.1 Spectral Methods

Monti et al. [105] designed Gaussian mixture model convolution (GMMConv) to graphs and meshes for learning task-specific representations. Yi et al. [173] introduced SynSpecCNN that parameterizes kernels in the spectral domain spanned by graph laplacian eigenbases for keypoint detection and part clustering. Ranjan et al. [123] proposed a spectral convolutional network that can capture non-linear variations of shapes for 3D face generation. Liu et al. [93] utilized convolution operating on 1-ring neighbors of each node in mesh for the classification of mild cognitive impairment and Alzheimer disease. Shu et al. [137] applied autoencoder to transform low-level features into high-level features of meshes and grouped the high-level features to co-segment 3D shapes.

2.4.2 Spectral-Free Methods

Litany et al. [92] proposed a variational graph convolutional autoencoder that learns hidden representations of meshes to complete partial shapes. Smith et al. [138] conducted GEOMetrics, a deep learning method based on graph convolution, for reconstructing mesh objects. Yao et al. [171] applied graph convolutional networks (GCNs) to learn motions from meshes for face reenactment. Kostrikov et al. [80] built Surface Networks, leveraging Dirac operator on meshes, for temporal prediction of mesh deformations and mesh generation. Wang et al. [158] proposed EdgeConv, a convolutional operation acting on point clouds, to handle high-level tasks, including classification and clustering. Wang et al. [155] introduced Pixel2Mesh that generates meshes from single RGB images based on GCNs. Hanocka et al. [56] established MeshCNN that utilizes geodesic connections among edges to process meshes for mesh classification and clustering.

In Chapter 4, I establish a GCN framework for effectively learning node and surface embedding for stream surface and isosurface.

CHAPTER 3

FLOWNET: A DEEP LEARNING FRAMEWORK FOR CLUSTERING AND SELECTION OF STREAMLINES AND STREAM SURFACES

3.1 FlowNet

Given a large set of streamlines or stream surfaces generated from a flow data set, I aim to identify a subset that best captures the underlying flow features and patterns. Instead of identifying the representatives directly, I opt to partition the input set into clusters and then select one from each cluster to form the representatives. A key question is how to learn the *feature descriptor* for a line or surface. I propose FlowNet design based on an *autoencoder* [11] that learns the feature representation using a deep neural net. I first voxelize and downsample each *object* (line or surface) into a 3D binary volume of an appropriate resolution, which will be the input to the autoencoder. The autoencoder trains the neural net and learns feature descriptors automatically. Instead of relying on labeled data for *supervised learning*, FlowNet applies the autoencoder for *unsupervised learning*, or more precisely, *self-supervised learning*. This eliminates the need to produce labeled data for training. Once the network converges, I apply t-SNE [151] to the feature descriptors for dimensionality reduction. I perform interactive clustering using DBSCAN [31] to identify the representatives. Finally, I design a visual interface for users to intuitively explore the line or surface collection and perform visual analysis and analytical reasoning.

3.1.1 Feature Descriptor Learning

Object voxelization. I define two representations of an object: *sequence* and *voxel* representations. The sequence representation of an object is a 1D vector, $\mathbf{s} = \{x_1, y_1, z_1, \cdots, x_n, y_n, z_n\}$,

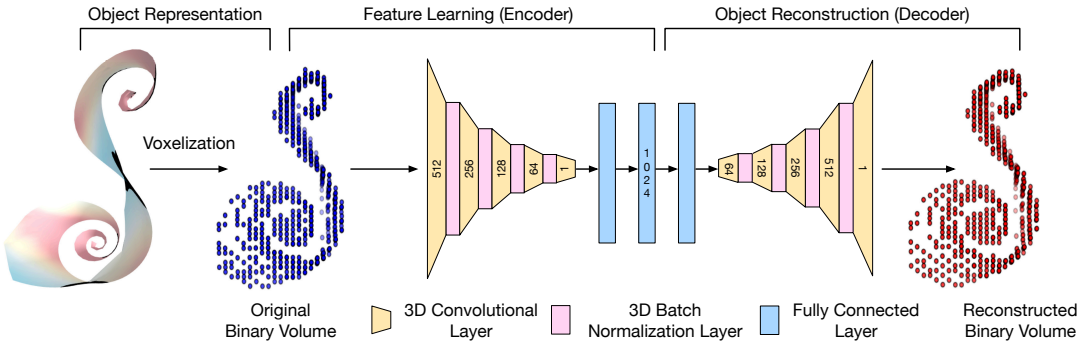
where (x_i, y_i, z_i) is a point on the object and n is the number of points. The voxel representation of an object is a volume $\mathbf{V}$ with size $L \times W \times H$. For streamlines, each object is a streamline represented by a sequence of points. For stream surfaces, each object is a stream surface where the sequence representation stores, line by line, the corresponding points following the streamline or timeline direction. I apply the rounding strategy so that each point on an object is mapped to its nearest voxel. If the voxel $\mathbf{V}[l_i, w_j, h_k]$ is occupied by the object, then the value of this voxel is 1 otherwise the value is 0.

Object voxelization transfers the sequence representation of an object into its voxel representation. The rule is that $\mathbf{V}[x_i, y_i, z_i] = 1$ for $i = 1$ to n and the remaining voxels are filled with 0. Due to the GPU memory constraint, given the original volume $\mathbf{V}$ of size $L \times W \times H$, I downsample it into a volume $\mathbf{V}^{'}$ of size $L^{'} \times W^{'} \times H^{'}$. I first calculate the downsamping ratio of each dimension: $x_r = L/L^{'}$, $y_r = W/W^{'}$, $z_r = H/H^{'}$. Then I set $\mathbf{V}^{'}[x_i^{'}, y_i^{'}, z_i^{'}] = 1$ for $i = 1$ to n, where $x_i^{'} = x_i/x_r$, $y_i^{'} = y_i/y_r$, $z_i^{'} = z_i/z_r$. The remaining voxels are set to 0.

CNN and autoencoder. As a class of deep, feed-forward artificial neural networks, a CNN performs the computation by *neurons*, which are organized into *layers* of different types: *convolutional* (CONV) and *fully-connected* (FC). The CONV layer detects local and non-linear combinations of features from the previous layer to capture important information from the raw data. The FC layer serves as further learning (from general to specific) of the input observation, combining local features into global features.

An autoencoder consists of two parts: *encoder* and *decoder*. The encoder takes an object as input and maps it to a feature descriptor. The decoder takes the feature descriptor as input and reconstructs the object. The basic autoencoder only consists of FC layers for unsupervised learning, which cannot ensure that the network learns a concise data representation and could impact its performance in reconstructing complex data such as 3D models. I can improve the reconstruction results by introducing CONV into the autoencoder. This is because there are always linear combinations of neurons in the FC layers

while CONV layers allow local and non-linear combinations of neurons, which enables the network to learn a complex data representation. Another advantage of using CONV layers is that it reduces the parameters to be learned through parameter sharing.



(a)

Figure 3.1. FlowNet for object feature learning. The input to FlowNet is the voxelized object representation. The network includes convolutional (CONV), batch normalization (BN), and fully-connected (FC) layers.

FlowNet architecture. As sketched in Figure 3.1, our FlowNet design includes two stages: *feature learning* (encoder) and *object reconstruction* (decoder). The first stage learns object features by non-linearly mapping each object representation to a feature descriptor (I use 1024 dimensions). Inspired by the work of Girdhar et al. [36], I design a CNN for feature learning and object reconstruction. Since there is no padding, I set the stride to 1 in all CONV layers. Moreover, I apply the *batch normalization* (BN) layer [68] to prevent the network from overfitting and gradient vanishing, while speeding up network training. The FC layers at the end enable the network to learn global features from local ones. The second stage is the inverse of the first. It reconstructs the object based on the feature descriptor learned. A loss function is used to indicate the error between the recon-

structed and original binary volume representations. FlowNet will adjust the parameters iteratively so that a more accurate feature representation can be learned.

Specifically, FlowNet takes a $L \times W \times H$ voxel representation of an object as input. The encoder consists of four CONV layers with BN added in between, one CONV layer without BN, followed by two FC layers. With five CONV layers and four BN layers, the decoder takes this embedded feature and maps it to a $L \times W \times H$ voxel grid. I apply the rectified linear unit (ReLU) [107] at the hidden layers and the sigmoid function at the output layer. Compared to other activation functions (e.g., tanh and sigmoid), ReLU can effectively avoid two major challenges in network training: gradient vanishing and explosion. Gradient vanishing happens when the gradient is close to zero in some hidden layers which prevents updating the parameters to their previous layers. Gradient explosion happens when the gradient is close to infinity which keeps the network from learning the structure of data. I train FlowNet with a binary cross-entropy loss on the final voxel output against the original voxel input. This loss qualifies the difference between the probability distributions of the true and predicted data. Other loss functions (e.g., mean squared error) will lead to a slower convergence of the network because they are prone to gradient vanishing [38]. The loss function of one training sample is defined as

$$\mathscr{L} = -\frac{1}{N}\sum_{i=1}^{N}\left[p_n \log \hat{p_n} + (1-p_n)\log(1-\hat{p_n})\right], \tag{3.1}$$

where p_n is the target probability (1 or 0) of a voxel being filled, $\hat{p_n}$ is the predicted probability obtained through FlowNet, and $N = L \times W \times H$. The total loss is the sum of losses for all training samples.

3.1.2 Dimensionality Reduction and Object Clustering

Exploring the feature descriptors generated from FlowNet for clustering and selection requires mapping these feature descriptors to a low-dimensional (e.g., 2D) space and then

grouping similar objects via clustering. The input to dimensionality reduction is the distance matrix recording the Euclidean distances between feature descriptors where each feature vector has been individually normalized [68] using L1-norm. After experimenting with three popular dimensionality reduction methods: t-SNE [151], MDS [81], and Isomap [145], and three widely used clustering algorithms: DBSCAN [31], k-means, and agglomerative clustering, I choose the combination of t-SNE and DBSCAN. All results presented for the rest of the chapter use this combination.

3.1.3 Interface and Interaction

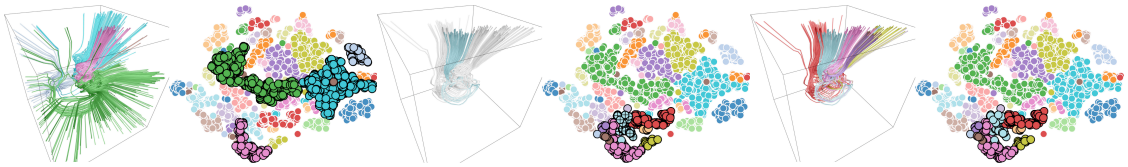
Our FlowNet interface consists of two views: the *volume view* and *projection view*. Both views are connected via brushing and linking: when users interact with one view, the other view will be automatically updated. The volume view displays the line or surface objects in the original 3D spatial domain and the projection view displays the objects as points in the abstract 2D space. I provide the following functions to explore these objects and their features descriptors:

Clustering. Our interface allows users to interactively tune the parameters of DBSCAN (e.g., the maximum distance between two feature descriptors and the minimum number of samples in one cluster) to generate the desired clustering results. To distinguish each cluster, I draw neighboring clusters using different colors. The selected cluster is highlighted with a black boundary and the volume view displays the corresponding objects. Users can also select multiple clusters simultaneously in the projection view and examine the relationships among them in the volume view. An example is shown in Figure 3.2 (a).

Representatives. To identify one representative from each cluster, I calculate the cluster's center as the data point where the sum of the Euclidean distances from this point to all the other points in the same cluster is the minimum. Users can interactively set the number of representatives. The volume view displays these representative objects, and the

projection view shows the clustering result with the centers highlighted.

Neighborhood. By computing the distance between the centers of two clusters as their inter-cluster distance, I allow users to "expand" one selected cluster to its neighborhood and conveniently explore the neighboring clusters. An example is shown in Figure 3.2 (b). To verify the similarities and differences among these neighboring clusters, users can examine these clusters one by one ordered by their distances to the selected cluster. Such an example is shown in Figure 3.2 (c).

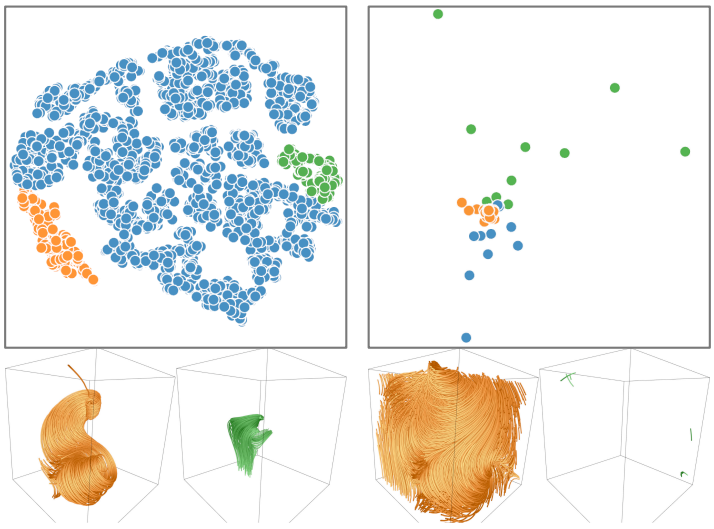


(a) (b) (c)

Figure 3.2: A sequence of cluster interactions in the t-SNE view and the linked volume view using the supernova data set: (a) choosing multiple clusters simultaneously, (b) expanding from one selected cluster highlighted at the bottom of the t-SNE view in (a) to its neighboring clusters, and (c) looping through each of these clusters (the focal cluster is highlighted with additional + signs). In (c), the remaining streamlines in the neighborhood are drawn in gray as the context.

3.1.4 Validation

Feature descriptor. To justify the need of deriving feature descriptors from streamlines, I compare the results generated using feature descriptors with FlowNet employed vs. direct use of binary volumes without FlowNet employed. The distance between two binary volumes is their summed, voxel-wise Euclidean distance. I project the streamline binary volumes with t-SNE for the five critical points data set, as shown in Figure 3.3 (b). The brushing and linking result shows that projecting binary volumes directly does not help to

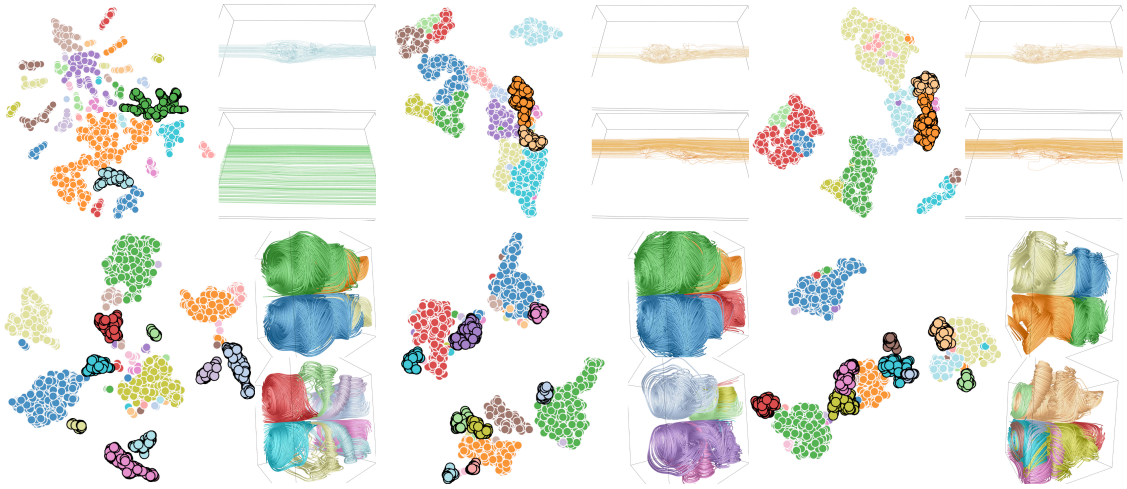


(a) (b)

Figure 3.3: Comparison of t-SNE projections of (a) feature descriptors and (b) binary volumes using the five critical points data set. Two point groups (orange and green) are selected via brushing and linking and their corresponding streamlines are shown. The unselected points are in blue.

reveal useful potential structures, for example, the streamlines around the critical regions that capture the main flow features, while projecting feature descriptors with t-SNE reveals these main features, as shown in Figure 3.3 (a). This comparison indicates that by extracting feature descriptors using the autoencoder, FlowNet can preserve structure exhibited by the streamline set rather than manufacturing structure by coincidence.

Distance measure. To verify the effectiveness of using the Euclidean distance between the corresponding feature descriptors for dimensionality reduction, I compare our distance measure against the mean of the closest point (MCP) distances between streamlines [176] and Hausdorff distance between streamlines [127] previously used to measure streamline similarity. I use t-SNE to project the data points and DBSCAN to group these points using the car flow and two swirls data sets, as shown in Figure 3.4. The result of the car flow data set shows that our distance measure can well separate the streamlines passing *through* the car (refer to the cyan cluster) from those passing *by* the car (refer to the green cluster), as shown in (a). Further brushing and linking shows that the surrounding small

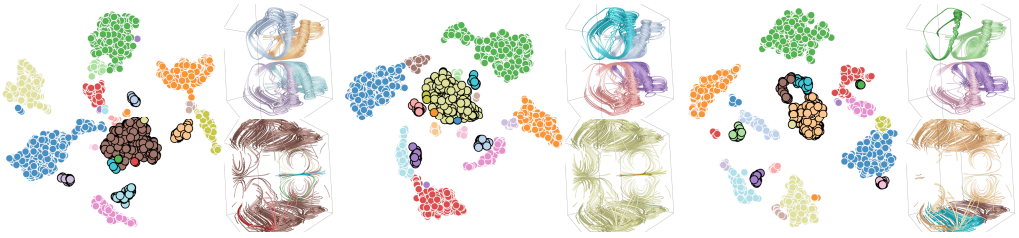


(a) (b) (c)

Figure 3.4: Comparison of different distance measures. (a) to (c) show FlowNet feature Euclidean distance, streamline MCP distance, and streamline Hausdorff distance. Top row: the car flow data set showing 35 clusters. Two streamline clusters are shown. Bottom row: the two swirls data set showing 36 clusters. Four largest streamline clusters are shown at the top-right. Eight selected neighboring streamline clusters are highlighted in the t-SNE view and shown at the bottom-right.

clusters correspond to streamlines located at the volume boundary. The other two distance measures, however, separate the streamlines passing through the car into different clusters, as shown in (b) and (c), which is not desirable. The result of the two swirls data set shows that all these three distance measures yield the four biggest streamline clusters which reveal the major structure of the two swirling patterns. However, our distance measure can better separate contextual streamlines from those streamlines in the biggest clusters. In addition, these contextual streamline clusters in (a) exhibit a better symmetry compared with those in (b) or (c).

Loss function measure. To verify the effectiveness of using binary cross-entropy for FlowNet training, I compare feature descriptors under three different loss functions: binary cross-entropy, mean squared error (MSE), and Dice loss [103]. I apply t-SNE projection and DBSCAN clustering for the two swirls data set, as shown in Figure 3.5. The clustering result shows that all these three loss functions can separate out the four biggest clusters.



(a) (b) (c)

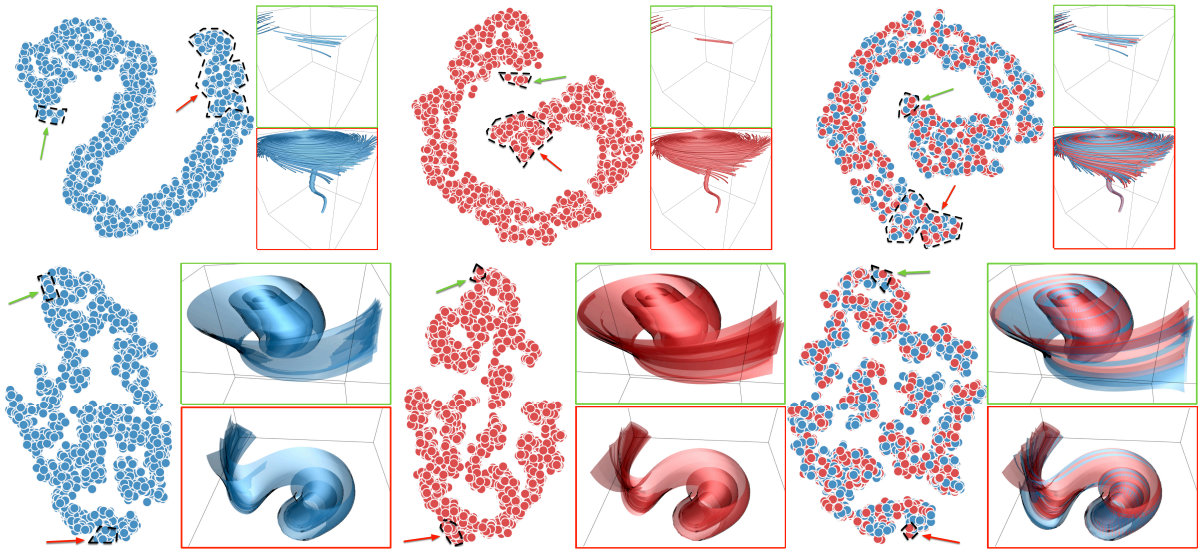
Figure 3.5: Comparison of feature descriptors under different loss functions using the two swirls data set. (a) to (c) show binary cross-entropy, F_1 score = 0.88, AMSE, F_1 score = 0.75, aDice loss[103], F_1 score = 0.84. All generate 25 clusters through DBSCAN. The selected streamline clusters are highlighted and shown in the t-SNE view.

However, compared to Dice loss, binary cross-entropy and MSE can discover the streamlines located at the volume boundary and therefore, better separate contextual streamlines. Moreover, using binary cross-entropy yields the highest F_1 score. Therefore, I choose binary cross-entropy as the loss function.

TABLE 3.1

THE DIMENSION OF EACH DATA SET AND RESPECTIVE KERNEL SIZE USED

Data Set	Original Dimension	Downsampled Dimension	Kernel Size
ABC	$51 \times 51 \times 51$	$51 \times 51 \times 51$	$3 \times 3 \times 3$
Bénard flow	$128 \times 32 \times 64$	$64 \times 16 \times 32$	$4 \times 1 \times 2$
car flow	$368 \times 234 \times 60$	$92 \times 59 \times 15$	$6 \times 4 \times 1$
computer room	$417 \times 345 \times 60$	$105 \times 87 \times 15$	$8 \times 6 \times 2$
crayfish	$322 \times 162 \times 119$	$81 \times 40 \times 30$	$4 \times 2 \times 2$
five critical pts	$51 \times 51 \times 51$	$51 \times 51 \times 51$	$3 \times 3 \times 3$
solar plume	$126 \times 126 \times 512$	$32 \times 32 \times 128$	$2 \times 2 \times 8$
square cylinder	$192 \times 64 \times 48$	$96 \times 32 \times 24$	$8 \times 3 \times 2$
supernova	$100 \times 100 \times 100$	$50 \times 50 \times 50$	$2 \times 2 \times 2$
tornado	$64 \times 64 \times 64$	$50 \times 50 \times 50$	$3 \times 3 \times 3$
two swirls	$64 \times 64 \times 64$	$32 \times 32 \times 32$	$4 \times 4 \times 4$



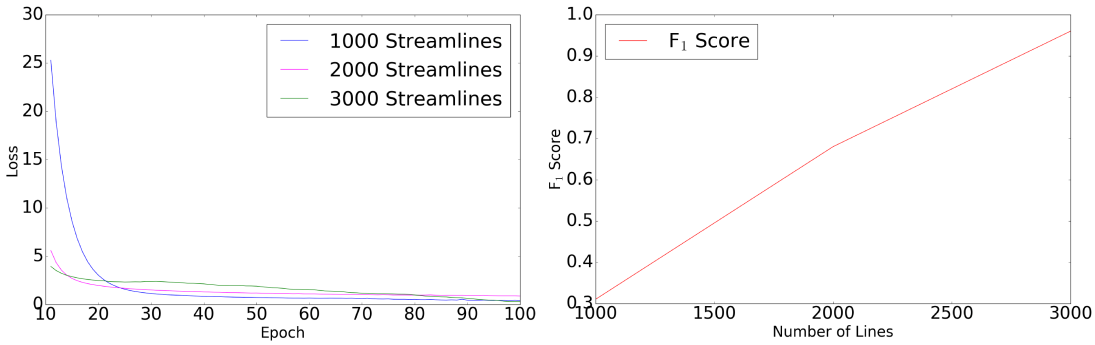
(a) training set only (b) test set only (c) training set + test set

Figure 3.6: Evaluation of FlowNet using the tornado streamline and stream surface data via brushing and linking. Two selected point groups and their corresponding streamlines or surfaces are shown. The training and test sets are in blue and red, respectively.

TABLE 3.2

PERFORMANCE COMPARISON AMONG DIFFERENT DATA SETS

Data Set	Training # Lines	Training F_1 Score	Testing # Lines	Testing F_1 Score	Training # Surfaces	Training F_1 Score	Testing # Surfaces	Testing F_1 Score
ABC	3,000	0.91	3,000	0.82	2,000	0.84	2,000	0.71
Bénard flow	3,000	0.87	3,000	0.80	2,000	0.84	2,000	0.79
car flow	3,000	0.81	3,000	0.69				
computer room	3,000	0.74	3,000	0.68	2,000	0.83	2,000	0.59
crayfish	3,000	0.86	3,000	0.78				
five critical pts	3,000	0.96	3,000	0.72	2,000	0.72	2,000	0.57
solar plume	4,000	0.83	4,000	0.76	1,000	0.84	1,000	0.57
square cylinder	3,000	0.84	3,000	0.72	2,000	0.91	2,000	0.86
supernova	3,000	0.86	3,000	0.75				
tornado	3,000	0.91	3,000	0.76	2,000	0.88	2,000	0.78
two swirls	3,000	0.88	3,000	0.75	2,000	0.91	2,000	0.81



(a) binary cross-entropy loss (b) F_1 score

Figure 3.7: Performance curves under different numbers of training streamlines for the five critical points data set.

Underfitting and overfitting. Two central challenges in machine learning are *underfitting* and *overfitting* [39]. Underfitting occurs when the model is not able to fit the training set. Overfitting occurs when the model fits the training set perfectly but fails to fit the test set. In Tables 3.1 and 3.2, I report for each data set, the sizes of training and test sets and their corresponding F_1 scores. Note that the test sets are randomly generated, in the same fashion as the training sets.

F_1 score is defined as

$$F_1 = \frac{2}{\frac{1}{\text{recall}} + \frac{1}{\text{precision}}} = \frac{2}{\frac{\text{FNs}+\text{TPs}}{\text{TPs}} + \frac{\text{FPs}+\text{TPs}}{\text{TPs}}}, \tag{3.2}$$

where TPs, FPs, and FNs stand for true positives, false positives, and false negatives, respectively. In our context, given a voxel in the binary volume, it is a true positive/false positive/false negative if the value of ground truth is 1/0/0 and the possibility predicted by FlowNet is greater/greater/less than 0.5. Ranging between 0 and 1, F_1 score defines the similarity between the original and the predicted objects. If F_1 score is closer to 1/0, it indicates that the predicted object is more/less similar to the original object.

The F_1 scores reported for training show that FlowNet is not underfitting. To visually demonstrate that FlowNet is not overfitting, I qualitatively compare the t-SNE views of the

training set and test set via separate projections, as shown in Figure 3.6 (a) and (b). Note that the embedding is with respect to the data points, not the underlying space. Although the orientations or spreads are different (which is due to the randomness of the t-SNE algorithm), both views share the similar global structure and local characteristics. Through brushing and linking, I can further verify that the trained model works as expected for the test set. I also experiment with projecting the training and test sets in the same view, as shown in Figure 3.6 (c). The interspersed points from both sets confirm that the trained model can map new, previously unseen inputs to appropriate feature vectors.

TABLE 3.3

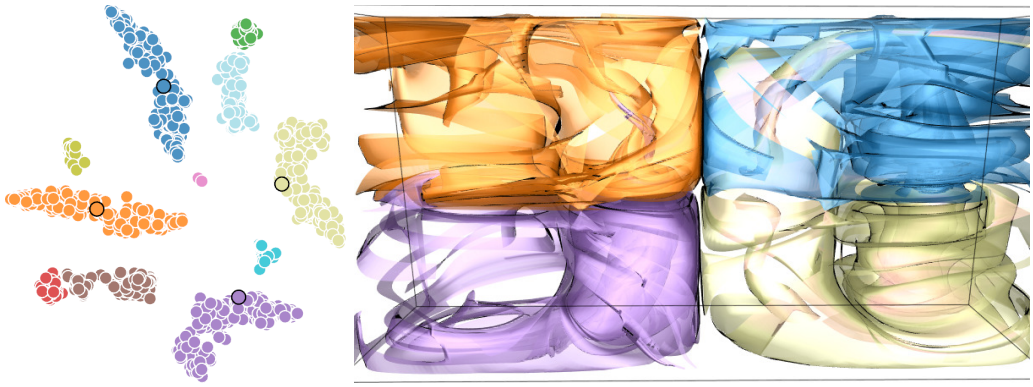
F_1 SCORES UNDER DIFFERENT TRAINING SAMPLES

Data Set	computer room	five critical pts	supernova	two swirls
# Lines	1,000	1,000	1,000	1,000
F_1 Score	0.36	0.31	0.38	0.35
# Lines	2,000	2,000	2,000	2,000
F_1 Score	0.62	0.68	0.69	0.67

3.2 Results and Discussion

3.2.1 Data Sets and Network Training

Data sets. I experimented with the list of data sets shown in Table 3.2. From top to bottom, these data sets are: the Arnold-Beltrami-Childress (ABC) incompressible flow which is an exact solution of Euler's equation [25], the liquid flow between two parallel



(a) (b)

Figure 3.8: Representative stream surface selection with t-SNE projection and DBSCAN clustering using the Bénard flow data set.

planes [161], the air flow around a car [99], the air flow in a computer room [160], the heat flow around a cooking crayfish [99], a synthesized flow field consisting of five critical points [172], the compressible downflow solar plume [124], the flow around a confined square cylinder [152], the flow of core-collapse supernovae [14], a procedurally generated tornado [22], and swirls resulting from wake vortices [99].

FlowNet training. I implemented FlowNet in PyTorch using an NVIDIA TITAN Xp 1080 GPU for network training. In the training process, I initialized all layers of FlowNet from scratch using $\mathcal{N}(0, 0.01)$ and applied the Adam optimizer [78] with learning rate 10^{-6} to update the parameters. I used the minibatch size of 1 and trained FlowNet with 100 epochs. For training efficiency, I follow a simple scheme to decide the training sample size. For streamlines, I initialize the training sample size with $1,000$ and train FlowNet. Then, I check the F_1 score after training FlowNet with 100 epochs. If the score is acceptable (e.g., larger than 0.7 based on our empirical experience), the training sample size is determined. Otherwise additional $1,000$ streamlines will be added to the training pool to retrain FlowNet. For stream surfaces, I initialize the training sample size with $1,000$, and if the training F_1 score is less than 0.7, additional 500 stream surfaces will be added to the training pool to retrain FlowNet. Based on this scheme, the size of the training set and the

corresponding kernel size are listed in Table 3.2. The testing F_1 scores for the computer room, five critical points, and solar plume surface data are relatively low (less than 0.6), mainly due to the complex flow features exhibited by these data sets.

In Figure 3.7, I report the binary cross-entropy losses and F_1 scores under different training samples for the five critical points data set. In (a), I can see that the binary cross-entropy loss converges fast as the number of training samples increases. In (b), I can see that the F_1 score significantly improves when using more streamlines in the training process. This indicates that the performance of FlowNet is highly related to the number of training samples used. Adding more streamlines to the training pool, FlowNet converges faster and F_1 score also improves. However, both benefits are at the expense of longer training time. Balancing between performance and training time, I choose an increment of $1,000$ streamlines for the training. In Table 3.3, I report F_1 scores under different training samples for different data sets. It is clear that using $1,000$ and $2,000$ streamlines cannot achieve a good performance (e.g., F_1 score is larger than 0.7), while the performance is acceptable if $3,000$ samples are used for training, as shown in Table 3.2. Based on this experiment, I conclude that using $3,000$ streamlines is enough to train FlowNet. For the solar plume data set, I use $4,000$ streamlines for training, leading to the F_1 score of 0.83 ($3,000$ streamlines only give the F_1 score of 0.67).

All streamlines are traced from seeds randomly placed in the domain. All stream surfaces are traced from random seeding curves following the binormal directions. Each seeding curve is generated by tracing in the binormal field with a random starting point and length [142]. For stream surfaces, I do not experiment with the car flow, crayfish, and supernova data sets, since the flow patterns in these data sets are either laminar or too complex to be effectively captured by the randomly-placed surfaces. The training time is mainly determined by the sizes of the kernel and training set. Our experiments show that the Bénard flow stream surface data require the lowest training time (15 minutes per epoch) while the computer room streamline data require the highest training time (90 minutes per

epoch). So training 100 epochs would take anywhere from one to seven days.

3.2.2 Results and Feature Understanding

Clustering and selection results. In Figure 3.2, Figures 3.4 to 3.6, and Figures 3.8 to 3.10, I show the results of clustering and selection of streamlines and stream surfaces. The brushing and linking results in Figure 3.3 (a) and Figure 3.6 show the good correspondence of neighboring points in the t-SNE view and neighboring lines or surfaces in the spatial view. This indicates that FlowNet feature vectors are a faithful representation of the underlying streamlines and stream surfaces in terms of their shapes and locations. It also shows that the t-SNE projection well preserves neighborhood information. The clustering results in Figures 3.2, 3.4, and 3.5 show that meaningful clustering and flexible exploration can be achieved using our method. In Figures 3.8 to 3.10, I show representative streamline and stream surface results. Unlike representative streamlines which are normally in the range of tens to hundreds, representative stream surfaces are typically within ten or up to tens. Therefore, I select representative streamlines automatically, while giving the option of a two-step process for representative stream surface selection. With this option, I first generate a certain number of representative stream surfaces and then let users pick a subset that strikes a balance between surface representativeness and domain coverage. In Figure 3.8, I show a set of representative surfaces following this two-step process. Four surfaces are selected from the t-SNE view showing 11 clusters. These four surfaces are the centroids of the largest four clusters. Figure 3.12 (a) and (c) show two other sets of representative surfaces based on the t-SNE view shown in Figure 3.8 (a). With this process, users are able to generate customized representative surface results. In Figure 3.10, the representative streamlines and surfaces show that I can generate a good representation of the flow fields with representative streamlines and surfaces at varying levels of detail. The appropriate number of representatives is determined empirically as users can make the adjustment interactively to generate desirable results.

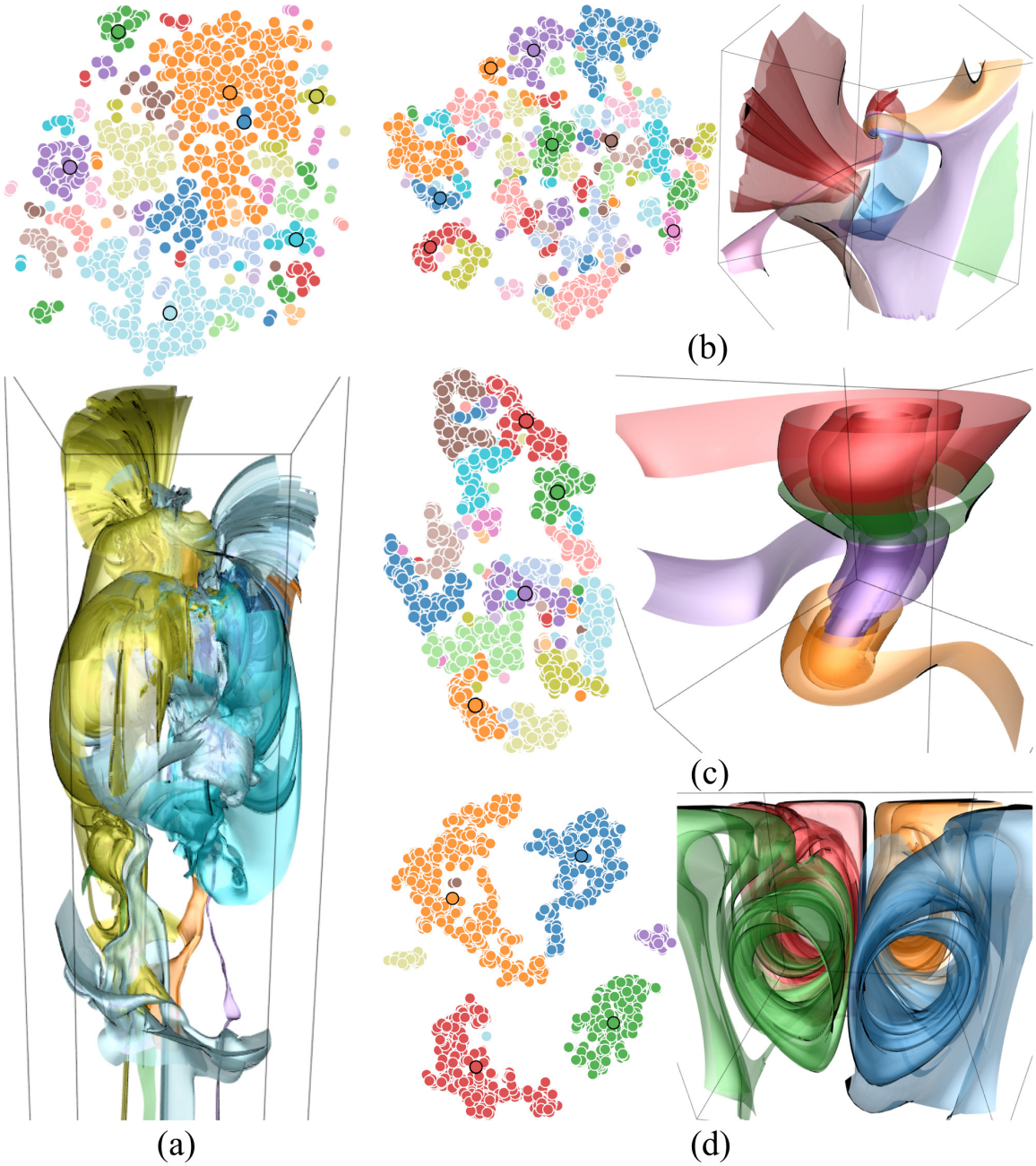


Figure 3.9: Representative stream surfaces of the solar plume, five critical points, tornado, and two swirls data sets. (a) to (d) show 7, 7, 4, and 4 representative surfaces, respectively.

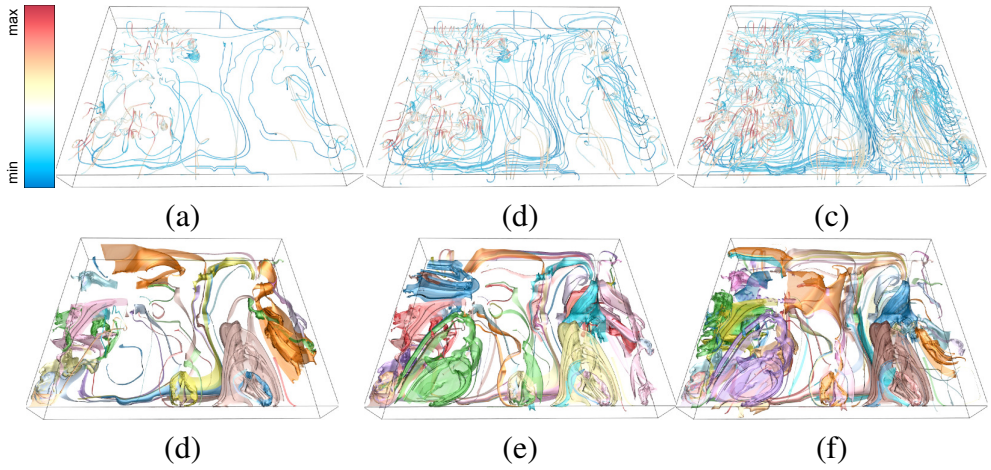
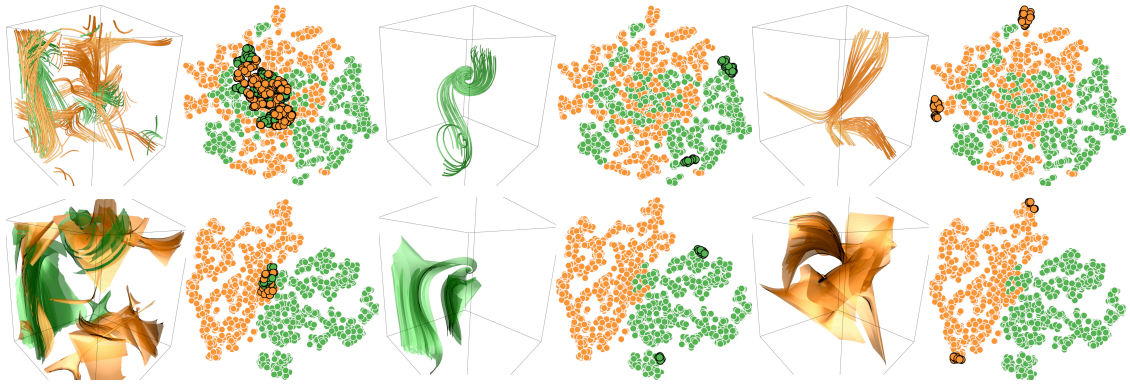


Figure 3.10: Representative streamlines and stream surfaces of the computer room data set. (a) to (c) show 70, 150, and 300 streamlines, respectively. (d) to (f) show 40, 60, and 80 stream surfaces, respectively. Velocity magnitudes are mapped to streamline colors.

Separation of cross-dataset features. I also experiment with the joint training of streamline and stream surface data drawn from different data sets. I select two data sets and use half of the training samples from each data set for joint training. The results are shown in Figure 3.11. I can see from the t-SNE view that the two data sets are largely separated in the projection as these two data sets contain very dissimilar flow features and patterns. The first row of Figure 3.11 shows that the overlapped points in the t-SNE view correspond to similar streamlines around the volume boundary while the separated points correspond to streamlines of distinct spatial locations and flow patterns. The stream surface results in the second row also confirm similar findings, although there is a less number of similar surfaces. This is mainly because stream surfaces are one dimension higher than streamlines. Using random seeding, it is less likely to generate similar stream surfaces from these two data sets. This experiment shows the potential of FlowNet in separating cross-dataset features and the possibility to generalize FlowNet to handle multiple data sets.



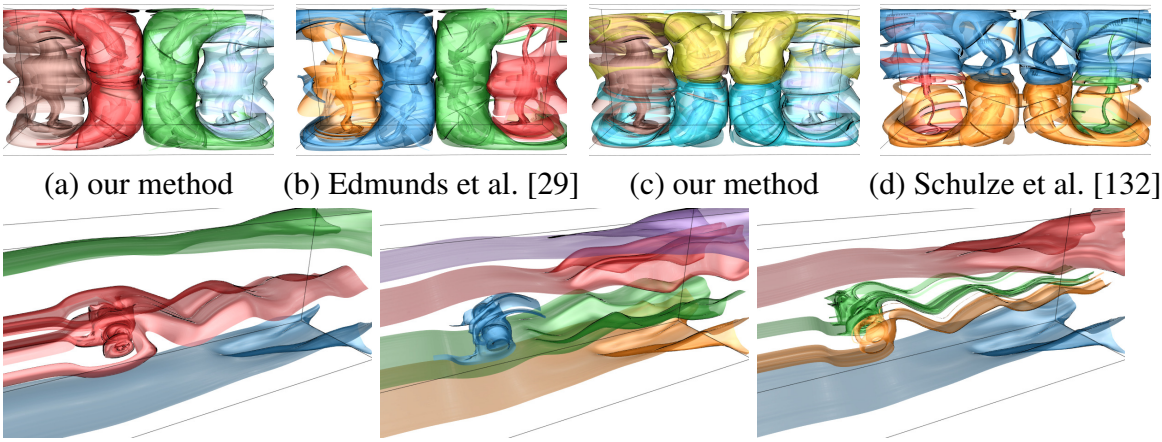
(a) (b) (c)

Figure 3.11: Joint training of the five critical points (green) and ABC (orange) streamline and stream surface data. (a) highlights where the two data sets overlap in the t-SNE view. (b) and (c) show an example of where the two data sets are separated in the t-SNE view.

3.2.3 Comparison against Existing Methods

Stream surface selection. In Figure 3.12, I compare our stream surface selection results against those generated from the feature-centered automatic surface seeding by Edmunds et al. [29] and the global selection of stream surfaces by Schulze et al. [132]. Both methods being compared are fully automatic, while our method provides the two-step process to users so that they can handpick representative stream surfaces. With this flexibility, for the Bénard flow data set, I are able to generate representative stream surface results similar to those generated by Edmunds et al. [29] (see (a) and (b)) and Schulze et al. [132] (see (c) and (d)). For the square cylinder data set, although the numbers of representative stream surfaces are not the same, the important surface features on the left are well captured by all three methods.

Streamline selection. In Figure 3.13, I compare our streamline selection results against those generated from the dual information channel based method of Tao et al. [143] and the entropy-based method of Xu et al. [170]. For fairness, each method produces the same number of streamlines. The information channel is built between the set of streamlines and a set of sample viewpoints. I generate the selection results using three criteria: $p(s)$ (streamline probability), $I(s;V)$ (streamline information), and REP (streamline representa-



(a) our method (b) Edmunds et al. [29] (c) our method (d) Schulze et al. [132]

(e) our method (f) Edmunds et al. [29] (g) Schulze et al. [132]

Figure 3.12: Top to bottom: comparison of surface selection results of the Bénard flow and square cylinder data sets using different methods. (a) to (d) show four stream surfaces each. (e) to (g) show three, five, and four stream surfaces, respectively.

tiveness). Our method and each of the three criteria of Tao et al. [143] select representatives from the same pool of streamlines. The entropy-based method generates streamlines iteratively guided by the conditional entropy between the original vector field and the field reconstructed from selected streamlines. By comparing the results side by side, I can observe that our method strikes a good balance between streamline informativeness and domain coverage, achieving comparable results with respect to those of REP. For the solar plume data set, our method yields the best domain coverage. For the five critical points data set, $p(s)$ fails to select surrounding streamlines which correspond to the saddle pattern, while our method and REP best capture the source at the center of the volume. For the tornado data set, $I(s;V)$ gives the best result by revealing the swirling pattern surrounding the vortex core at the bottom. For the two swirls data set, our method strikes the best balance between domain coverage and feature highlighting. Overall, I feel that using FlowNet features generates competitive streamline selection results comparing to these state-of-the-art solutions.

To quantitatively evaluate the quality of selected streamlines, I use them to reconstruct the vector field $\mathbf{V}'$ through gradient vector flow [168]. I follow the work of Tao et al. [143]

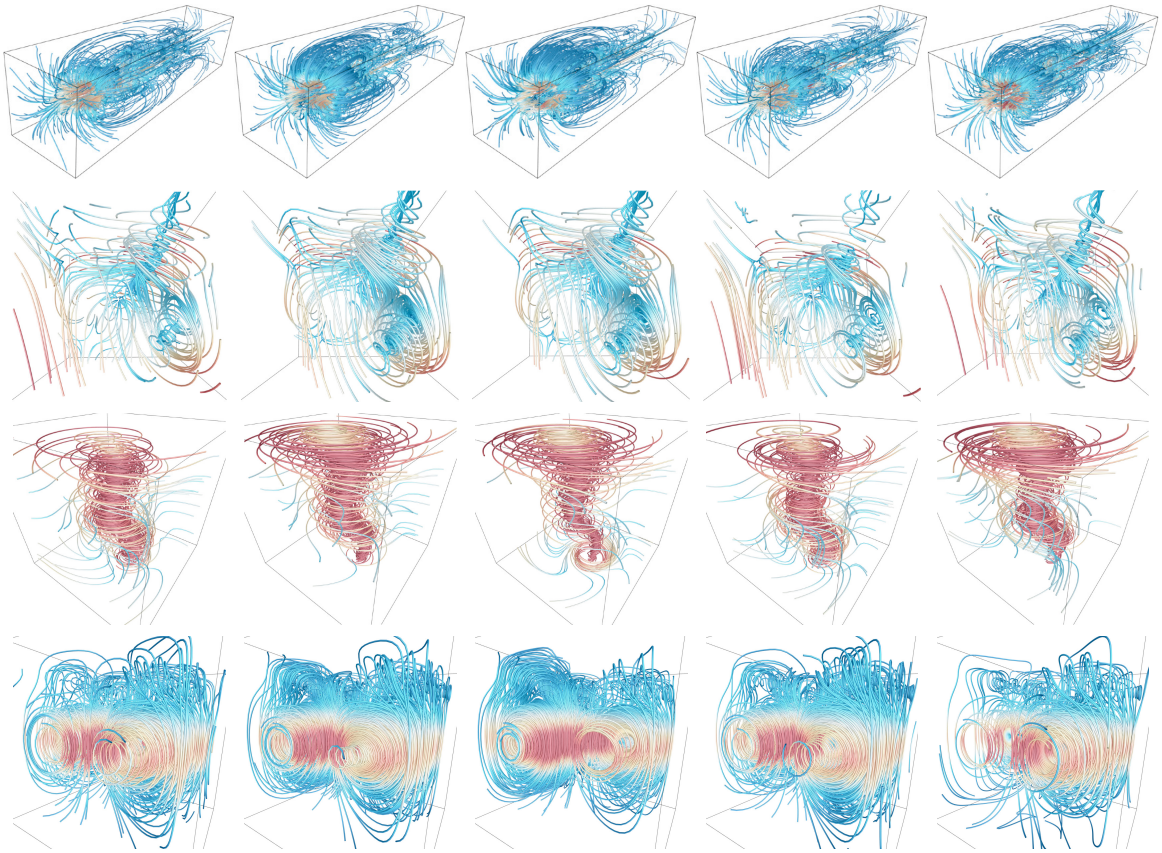
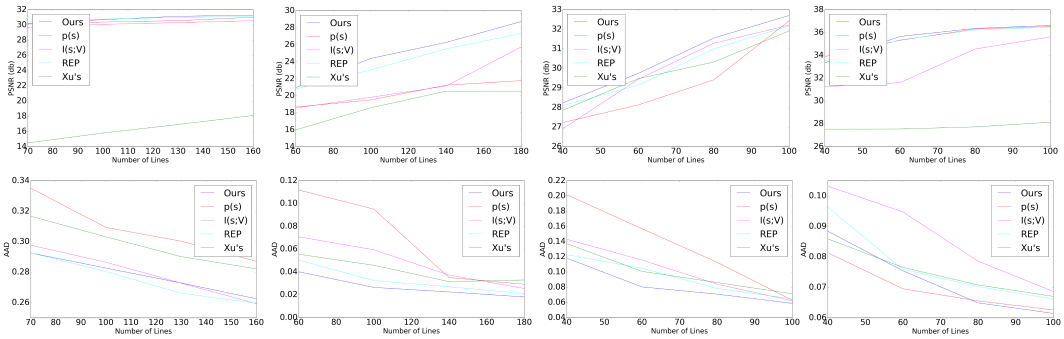


Figure 3.13: Top to bottom: comparison of streamline selection results of the solar plume, five critical points, tornado, and two swirls data sets using different methods. (a) to (e) show our method, $p(s)$ Tao et al. [143], $I(s;V)$ Tao et al. [143], REP Tao et al. [143], and Xu et al. [170]. All methods show the same number of streamlines: 100, 140, 60, and 80, respectively.

to initialize $\mathbf{V}'$ and iteratively refine $\mathbf{V}'$ using the generalized diffusion equations. After that, I compute the peak signal-to-noise ratio (PSNR) and average angle difference (AAD) of $\mathbf{V}'$ with respect to the original vector field $\mathbf{V}$. For AAD, I calculate the angle difference between the original and reconstructed vectors for all voxels and then get the average. I normalize the error to $[0,1]$ by dividing the AAD by π. A method is the best if it leads to the largest PSNR and the lowest AAD. In Table 3.4, under a given setting for the number of streamlines, I can see that our method achieves the highest PSNR and lowest AAD except for the solar plume data set. For that data set, REP achieves the best quality, while



(a) solar plume (b) five critical points (c) tornado (d) two swirls

Figure 3.14: Comparison of PSNR (top row) and AAD (bottom row) of reconstructed vector fields under different streamline selection methods and different numbers of training streamlines.

our method is the second best in terms of both PSNR and AAD. More extensive quality comparison results are given in Figure 3.14. I can see that in general, our method is the best under different streamline selection methods and different numbers of training streamlines.

TABLE 3.4

PERFORMANCE COMPARISON OF RECONSTRUCTED VECTOR FIELDS

UNDER DIFFERENT STREAMLINE SELECTION METHODS

		PSNR (dB)					AAD				
Data Set	# Lines	Ours	$p(s)$	$I(s;V)$	REP	Xu's	Ours	$p(s)$	$I(s;V)$	REP	Xu's
crayfish	70	**30.94**	30.84	30.91	30.02	28.97	**0.102**	0.105	0.103	0.116	0.144
solar plume	100	30.68	30.37	30.07	**30.75**	15.78	0.283	0.309	0.286	**0.280**	0.303
five critical pts	140	**26.25**	21.23	21.13	25.50	20.16	**0.023**	0.031	0.036	0.026	0.031
tornado	60	**29.74**	28.12	29.44	29.13	28.30	**0.080**	0.167	0.116	0.105	0.101
two swirls	80	**36.35**	36.30	34.55	36.21	27.72	**0.065**	0.066	0.079	0.070	0.071

It is the clear winner for the five critical points and tornado data sets. For the crayfish data set, our method starts with the worst quality but ends with the best. For the solar plume data set, our method achieves the highest PSNRs very similar to REP, while loses to $I(s;V)$ and REP by a small margin in terms of AAD when the number of streamlines is larger than 140. For the two swirls data set, our method achieves the highest PNSRs very similar to $p(s)$ and REP, while the lowest AAD when the number of streamlines is larger than 80. I conclude that our method actually performs almost the best quantitatively compared to $p(s)$, $I(s;V)$, REP, and Xu's method.

3.3 Conclusions

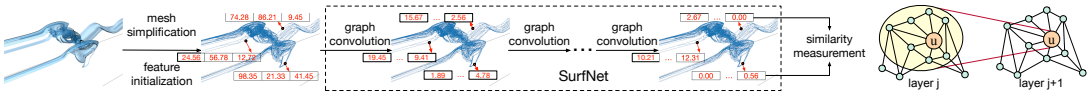
I have presented FlowNet, a novel approach for clustering and selection of streamlines and stream surfaces. Based on the encoder-decoder, FlowNet is able to learn latent features of streamlines and stream surfaces within a single framework in an unsupervised manner, which distinguishes itself from all previous works which have to solve them separately and explicitly utilize handcrafted features. These latent features encode the shape and location information of objects rather than the physical flow information. I then project the resulting feature vectors into a low-dimensional space, which lends itself to a visual mapping and interface for user interaction. Brushing and linking yields meaningful clustering and selection results. The line and surface clusters generated from FlowNet capture both spatial proximity and/or geometric similarity. I validate FlowNet using the network learned from the training set to examine the test set, and compare the results using FlowNet-trained features against those using other state-of-the-art methods.

CHAPTER 4

SURFNET: LEARNING SURFACE REPRESENTATIONS VIA GRAPH CONVOLUTIONAL NETWORK

4.1 SurfNet

Given a large set of stream surfaces or isosurfaces generated from a vector or scalar field data set, we propose two aims: clustering and selection. For clustering, we aim to partition every stream surface or isosurface into several parts so that these parts exhibit different patterns. For selection, a subset of surfaces best covering the underlying features and patterns needs to be identified. Instead of identifying these surfaces directly, we group the input set into clusters and select the representatives from these clusters. A key question is how to generate node features for a surface in an unsupervised manner. We propose SurfNet, a GCN that learns node features through graph convolution. An overview of SurfNet is sketched in Figure 4.1 (a). We first simplify the surfaces using a mesh simplification algorithm [34] as needed. Then, we transform every simplified surface into an undirected unweighted graph, which will be the input to SurfNet. We initialize the node features using their positional information. SurfNet learns node features automatically by aggregating their neighborhoods. To optimize SurfNet, we compute a node similarity matrix, and based on this matrix, SurfNet will update learnable parameters through gradient descent. Once SurfNet converges, these node features represent each node's information or the entire surface's information. We then apply t-SNE [151] to node features for projecting to a low-dimensional space and leverage DBSCAN [31] to identify the clustering or the representatives based on interactive clustering. Finally, users can explore the surface(s) and perform visual analysis and analytical reasoning through a visual interface.



(a) (b)

Figure 4.1: (a) SurfNet for node embedding learning. The input to SurfNet is the simplified surface. Several graph convolutions are leveraged to learn node embeddings. Finally, a similarity measure is applied to optimize SurfNet. (b) An example of embedding propagation in GCN. The latent embedding for node u at layer $j+1$ is aggregated from its previous embedding and immediate neighbors at layer j.

4.1.1 Notation

Formally, we define a surface (mesh) as a graph $G = (V,E)$, where V is the set of all nodes (e.g., sample tracing points along both streamline and timeline directions on a stream surface or triangle vertices on an isosurface) and E is the set of all edges (e.g., edges connecting sample tracing points on a stream surface or triangle edges on an isosurface). Each node $u \in V$ has a set of associated feature values, denoted as $\mathbf{F}_u$. $\mathbf{F}_u$ could denote various pieces of information (e.g., position, normal, velocity) of the corresponding node u. $\mathcal{N}(u)$ denotes the neighborhood of u. In addition, $\mathbf{G} = \{G_1, G_2, \ldots, G_n\}$ is a set of graphs and $\mathbf{F}^{G_i}$ represents the feature descriptor of graph G_i.

4.1.2 Node Embedding

Given a surface, SurfNet aims to generate a set of node features, and each node feature presents rich spatial and geometric information of the corresponding node on the surface. To this end, SurfNet needs to satisfy the two major properties as suggested by Bai et al. [4]:

- **Inductivity.** SurfNet should learn a unified mapping function so that it can be directly applied to any unseen surface to generate the corresponding node features.
- **Permutation invariance.** A different adjacency matrix can represent the same surface by permuting the order of nodes, and SurfNet should be insensitive to such permutations.

Across different node embedding models, neighborhood aggregation methods based on

GCN are permutation-invariant and inductive. This is because the core operation, graph convolution, updates a node's representation by aggregating the node and its neighbors' features. Since the aggregation function treats a node's neighbors as a set, the order does not affect the final embedding result.

Node aggregation. To merge information on a node u, we choose a topology-based aggregator [174], as sketched in Figure 4.1 (b). The aggregator works as follows. For each node u on a surface, all nodes v, where $v \in \mathcal{N}(u)$, are first identified. Then, a multi-perceptron layer is leveraged to transform the node representation $\mathbf{F}_v$ into a new node representation through an aggregation function (e.g., averaging or summation). This aggregation function merges representations $\mathbf{F}_u$ and $\mathbf{F}_v$, where $v \in \mathcal{N}(u)$. After that, we apply an activation function to the transformed embedding to increase the capability of capturing nonlinear behaviors. Finally, a representation of u consisting of the node and its local neighborhood's information is produced.

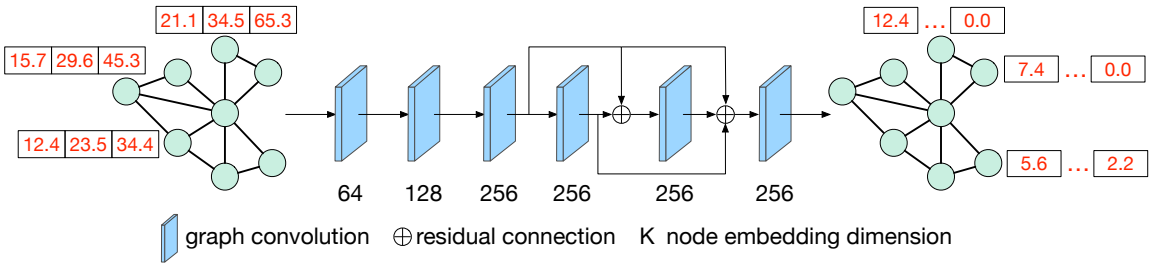


Figure 4.2. The architecture of SurfNet. SurfNet contains six graph convolution layers and the last four graph convolutions are bridged by residual connection.

Network architecture. As sketched in Figure 4.2, following Ying et al. [174], after applying one graph convolution to a surface, we update the feature representation of each

surface node u. We employ several graph convolutions to aggregate enough information from u's neighbors. Specifically, SurfNet contains six graph convolution layers. These six layers embed the features into 64, 128, 256, 256, 256, and 256 dimensions, respectively. Moreover, we leverage residual connection [58] to bridge the last four graph convolution layers, mitigating the impact of vanishing gradients and accelerating convergence [169]. We apply the rectified linear unit (ReLU) [107] as the activation function after each graph convolution layer. Note that the initial representations (i.e., the input to the first layer) could be the original information of the surface nodes (e.g., position, normal). The detailed parameters are listed in Table 4.1. Note that we increase the embedding dimension in network design so that the learned features have enough information to represent nodes.

TABLE 4.1

NETWORK PARAMETER DETAILS OF SURFNET

operation	# input neurons	# output neurons
graph Conv	3	64
graph Conv	64	128
graph Conv	128	256
graph Conv	256	256
graph Conv	256	256
graph Conv	256	256

4.1.3 Loss Function

To optimize SurfNet, we investigate several loss functions for measuring node similarity in the feature space.

- Optimization based adjacency matrix [3]:

$$\mathscr{L} = \sum_{i=1}^{n} \sum_{u \in V_i} \sum_{v \in V_i} (\mathbf{A}_i(u,v) - \mathbf{F}_u^T \mathbf{F}_v)^2, \tag{4.1}$$

  where n is the number of graph samples (i.e., $\{G_1, G_2, \ldots, G_n\}$) used for training, V_i is the node set of G_i, and $\mathbf{A}_i$ is the adjacency matrix of G_i.
- Optimization based on random walk [44]:

$$\mathscr{L} = \sum_{i=1}^{n} \sum_{u \in V_i} \left[-\log(\sigma(\mathbf{F}_u^T \mathbf{F}_v)) - \sum_{k \in P_u} \log(\sigma(\mathbf{F}_u^T \mathbf{F}_k)) \right], \tag{4.2}$$

  where v is a node that is near u on a fix-length random walk (e.g., 10), σ is the sigmoid function, and P_u is a negative sampling node set of node u.
- Optimization based on node distance: Inspired by Corso et al. [21], which embeds biological sequences by reducing the difference between the sequences' edit distance and the distance between the learned embeddings, we design a novel self-supervised loss as follows

$$\mathscr{L} = \sum_{i=1}^{n} \sum_{u \in V_i} \sum_{v \in V_i} (\mathscr{D}(u,v) - d(\mathbf{F}_u, \mathbf{F}_v))^2, \tag{4.3}$$

  where $d(\cdot,\cdot)$ denotes the distance measure (such as L_2 norm) in the feature space and $\mathscr{D}(u,v)$ is a distance metric between nodes u and v (e.g., position, normal, or geodesic similarity).

Loss analysis. To investigate the effectiveness of various losses, we train SurfNet with different objective functions (i.e., adjacency matrix, random walk, Euclidean, normal, velocity, and geodesic distances) using the five critical points data set. The results are given in Figure 4.3. Node clustering results are not satisfactory using the adjacency matrix, random walk, normal, and velocity distances. For example, these loss functions cannot isolate the spirals on this surface. Euclidean distance can detect the spiral at the top of the surface (i.e., the yellow part) but fails to discover the other one. The geodesic distance can separate the two spirals at both ends (the yellow and blue parts). Therefore, in the chpater, we choose geodesic distance as the objective function for SurfNet optimization.

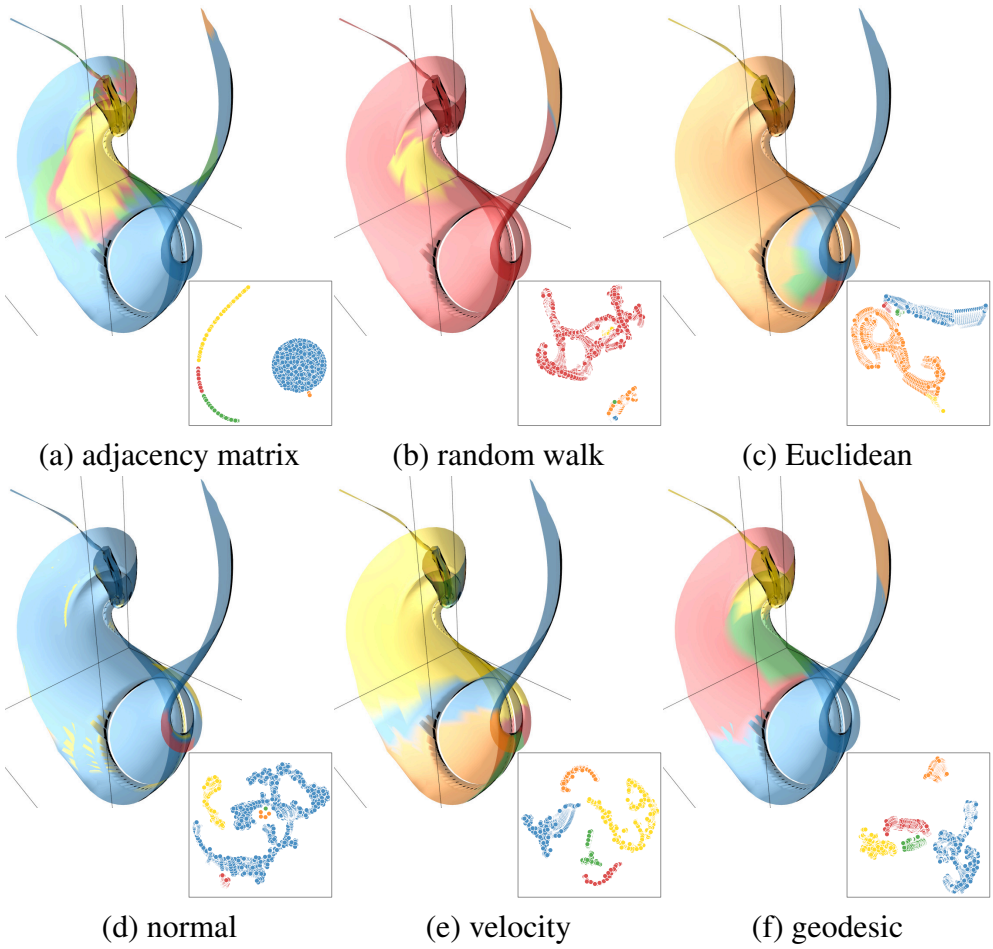


Figure 4.3: SurfNet node clustering results of a stream surface under different loss functions using the five critical point data set.

We further reason these losses during network optimization. As illustrated with an example in Figure 4.4, Euclidean distance deems red and yellow nodes closer than red and purple nodes. This contrasts the observation that red and purple nodes should be more similar since they come from the same surface branch, while red and yellow nodes are less similar as they reside in two different branches. For adjacency matrix and random walk, both determine that the red node is *equally* distant from the green and yellow ones. The red node is disconnected from the yellow or green node in the adjacency matrix. The red node cannot reach the yellow or green node through multiple random walks. Only the loss function based on the geodesic distance reflects the correct similarity order.

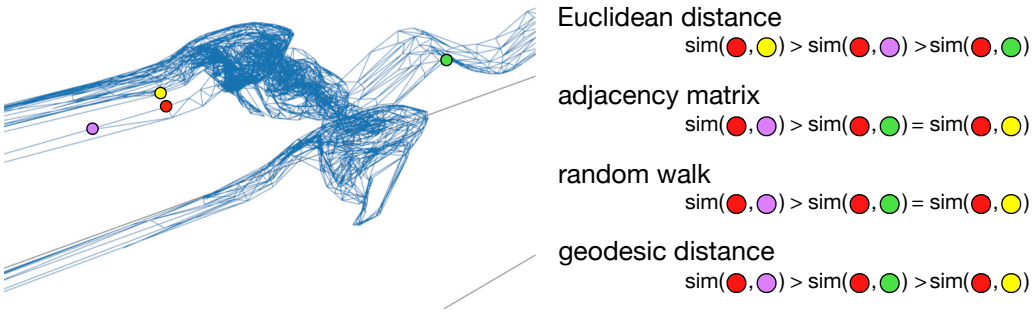
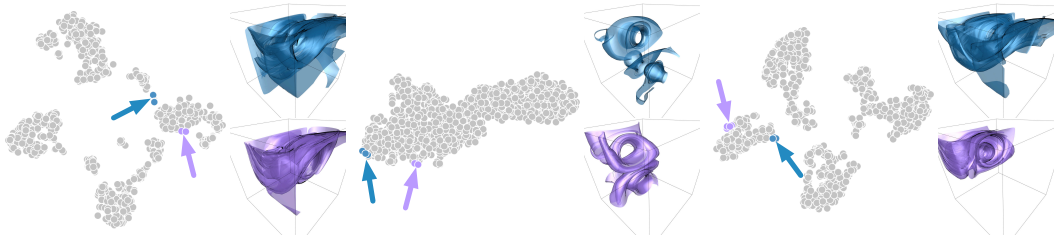


Figure 4.4: Comparison of different loss functions. The correct similarity order is sim(●,○)>sim(●,●)>sim(●,○). This is because red, purple, and green nodes are on the same surface branch while red and yellow nodes are on two different branches. In addition, red and purple nodes are closer on the surface than red and green nodes.



(a) CD (b) JSD (c) SurfNet

Figure 4.5: Evaluation of different surface distance metrics using the two swirls data set via brushing and linking. The unselected nodes are colored in gray.

4.1.4 Surface Embedding

For surface G_i, given a set of node embedding features $\mathbf{F} = \{\mathbf{F}_1, \mathbf{F}_2, \cdots, \mathbf{F}_{|V_i|}\}$, we compute the corresponding surface embedding feature as

$$\mathbf{F}^{G_i} = \frac{1}{|V_i|} \sum_{u \in V_i} \mathbf{F}_u, \tag{4.4}$$

where $|V_i|$ is the number of nodes in G_i.

Similar to Tkachev et al. [148], the distance between surfaces G_i and G_j is defined as

$$\mathscr{D}(G_i, G_j) = ||\mathbf{F}^{G_i} - \mathbf{F}^{G_j}||_2, \tag{4.5}$$

where $||\cdot||_2$ is L_2 norm.

To demonstrate the effectiveness of this surface distance metric, we compare this metric with two traditional surface distance metrics, i.e., chamfer distance (CD) [6] and Jensen-Shannon divergence (JSD) [88]. A comparison is shown in Figure 4.5. In the projection space generated by each distance metric, we brush two parts and display the corresponding stream surfaces. For CD, similar surfaces are not placed in close locations. For JSD, the projection cannot show any relationship among those surfaces. For SurfNet, we can observe that the surfaces can be classified into four groups in general and similar surfaces can be placed together.

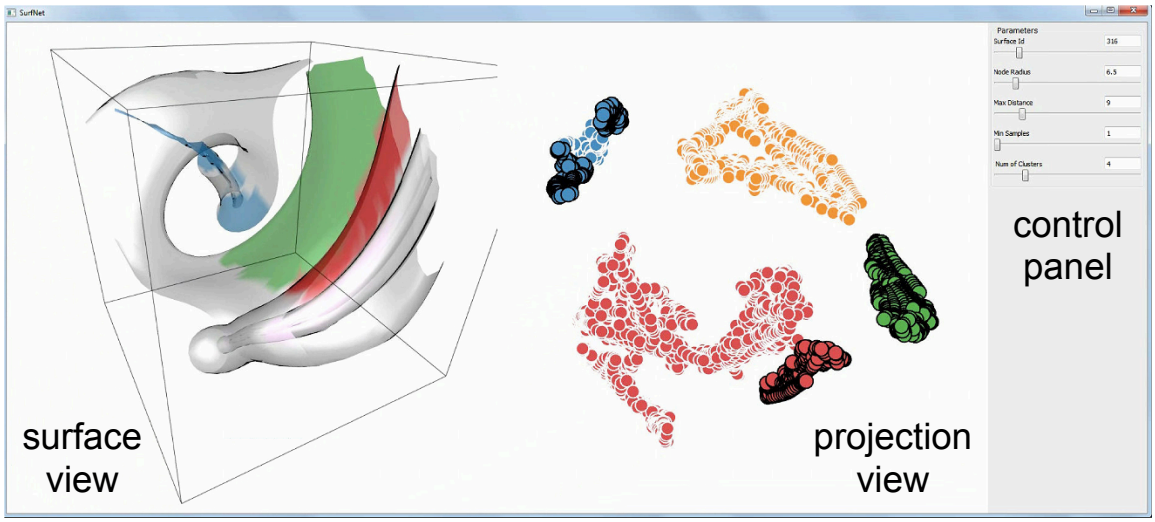


Figure 4.6: The visual interface of SurfNet. In this example, the selected nodes are highlighted in both views. The unhighlighted surface parts are colored with light gray in the surface view.

4.1.5 Interface and Interaction

The screenshot in Figure 4.6 shows that our SurfNet interface includes two views: *surface view* and *projection view*. Brushing and linking are used to connect both views. Surface(s) in the original 3D space is shown in the surface view, and the projected points in the 2D space are displayed in the projection view. Note that for node clustering, each point in the projection view represents a *node* on the surface (*node embedding*). For representative selection, each point represents a *surface* (*surface embedding*). The following interactive functions are supported to explore the clustering and selection results.

- **Clustering.** The hyperparameters of DBSCAN (i.e., the maximum distance between two node features in the 2D space and the minimum number of samples in one cluster) can be tuned by users to produce different clustering results. Neighboring clusters are drawn using different colors for differentiation. A black boundary is added to the selected cluster for highlighting, and the corresponding surface parts or surfaces are displayed in the surface view. Multiple clusters in the projection view can be chosen simultaneously by users. The relationships among them are examined in the surface view. The unselected ones are colored with light gray.
- **Representatives.** To select the representative from one cluster, we follow FlowNet [51] and define a cluster's center as the point where the average distance is the shortest between this point and all the other points in this cluster. The number of representatives can be manually changed based on user preferences. The selected representatives are displayed in the surface view, and the corresponding points are shown in the projection view.
- **Neighborhood.** Following FlowNet [51] , we define the distance between two clusters as the distance between their centers. Users can select one cluster and expand to its neighborhood to explore the neighboring clusters. Users can also go through these clusters according to their distances to the selected one. They can check the similarities and differences among the clusters in the neighborhood.

4.2 Results and Discussion

4.2.1 Data Sets and Network Training

We experimented with vector and scalar data sets shown in Tables 4.2 and 4.3, respectively. We used two multivariate data sets (combustion and ionization), and the rest of the data sets have a single variable. We implemented SurfNet using PyTorch [113]

TABLE 4.2

DIMENSION AND TRAINING EPOCHS OF VECTOR DATA SETS

data set	dimension ($x \times y \times z$)	epochs
Bénard flow	$128 \times 32 \times 64$	200
five critical points	$51 \times 51 \times 51$	200
solar plume	$126 \times 126 \times 512$	200
square cylinder	$192 \times 64 \times 48$	200
tornado	$64 \times 64 \times 64$	200
two swirls	$64 \times 64 \times 64$	200

and DGL [159]. The training and inference were run on an NVIDIA GTX 1080 Ti GPU. In terms of optimization, we initialized SurfNet parameters following the suggestion of He et al. [57] and employed the Adam optimizer [78] for parameter updates ($\beta_1 = 0.9$, $\beta_2 = 0.999$). We set one training sample per mini-batch and the learning rate to 10^{-4}. For stream surfaces, we generated $2,000$ surfaces. We used $1,000$ surfaces for training and the other $1,000$ for inference. For isosurfaces, we uniformly selected 256 isovalues for training and sampled other isovalues for inference. We determined all these hyperparameters empirically.

To evaluate SurfNet, we analyze different hyperparameter settings, including mesh simplification, network depth, training stability, embedding strategy, embedding dimension, feature initialization, and training samples.

4.2.2 Node Clustering

Baselines. For node clustering, we compare SurfNet against both spectral and spectral-free methods:

TABLE 4.3

DIMENSION AND TRAINING EPOCHS OF SCALAR DATA SETS

data set	variable	dimension ($x \times y \times z$)	epochs
combustion	HR, MF, YOH	$240 \times 360 \times 30$	100
ionization	H, He, PD	$600 \times 248 \times 248$	100
bonsai	Intensity	$204 \times 204 \times 204$	100
lobster	Intensity	$301 \times 324 \times 56$	300

- GMMConv [105] is a spectral method. It learns d-dimensional node embedding through Gaussian mixture CNNs.
- EdgeConv [158] is a spectral-free method. It encodes each node into a d-dimensional vector by aggregating its nearest neighbors.

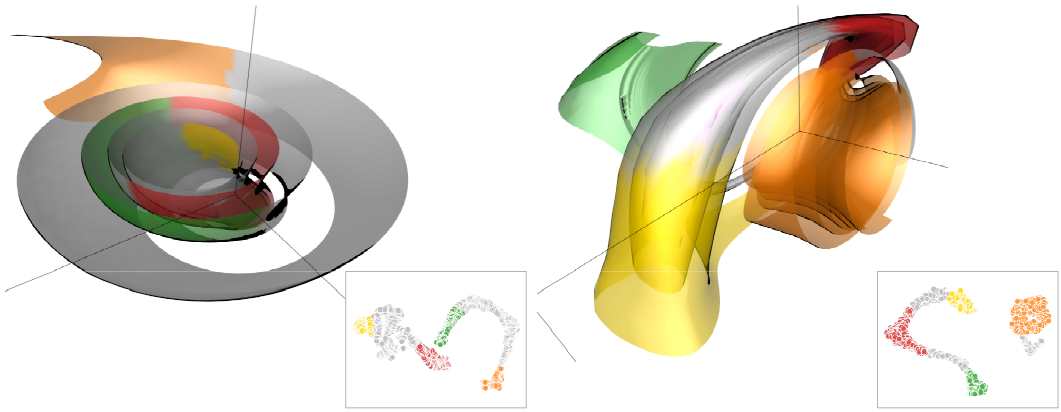
Note that other works [80, 92, 93, 123, 137, 138, 155, 171, 173] are not suitable for learning node embedding without supervision since these architectures are tailored for mesh generation, classification, and segmentation tasks. Additionally, MeshCNN [56] aims to learn edge embedding, which cannot be optimized using the node-based loss.

For a fair comparison, we apply the same settings (i.e., the loss function, number of training samples, optimizer, and epochs) for optimizing GMMConv, EdgeConv, and SurfNet.

Similar to other node embedding approaches [2, 66, 71, 79], the 2D t-SNE projection cannot completely represent relationships among different nodes.

Node feature validation. To verify the effectiveness of our node embedding, we apply t-SNE to project node embeddings to a 2D space and then perform brushing and linking. The results are shown in Figure 4.7. The t-SNE projection conveys the node's positional information and potentially the surface's structural information. For example, for the tornado (Figure 4.7 (a)), the orange points at the bottom-right corner of the projection view correspond to the tail of the surface, and the green and red points correspond to the spiral.

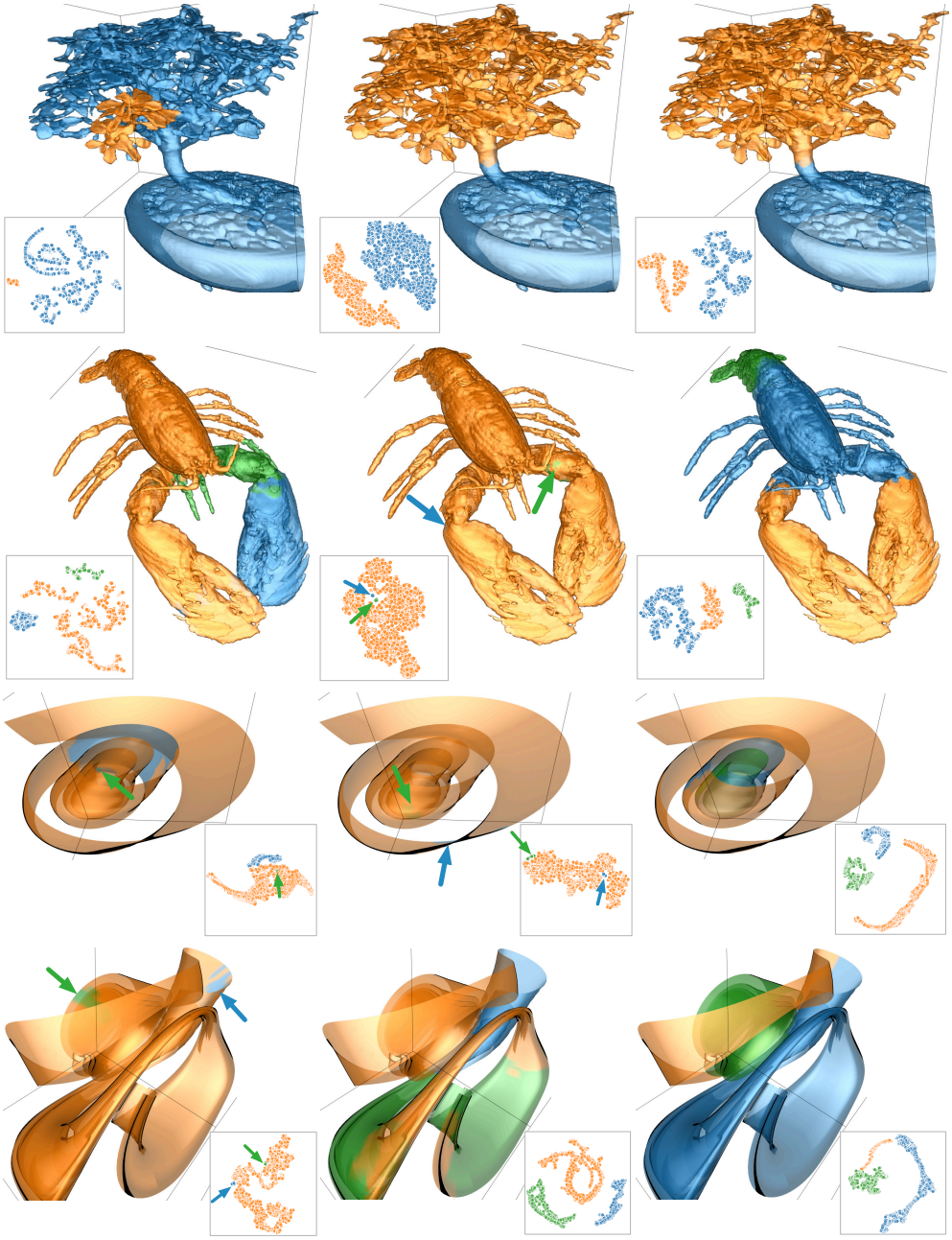
Likewise, for the two swirls (Figure 4.7 (b)), the orange points in the projection view exhibit a swirling pattern, which corresponds to the orange part of the surface. These brushing and linking results demonstrate the meaningfulness of the learned features, indicating that they can capture the nodes' neighborhoods and their positional information.



(a) tornado (b) two swirls

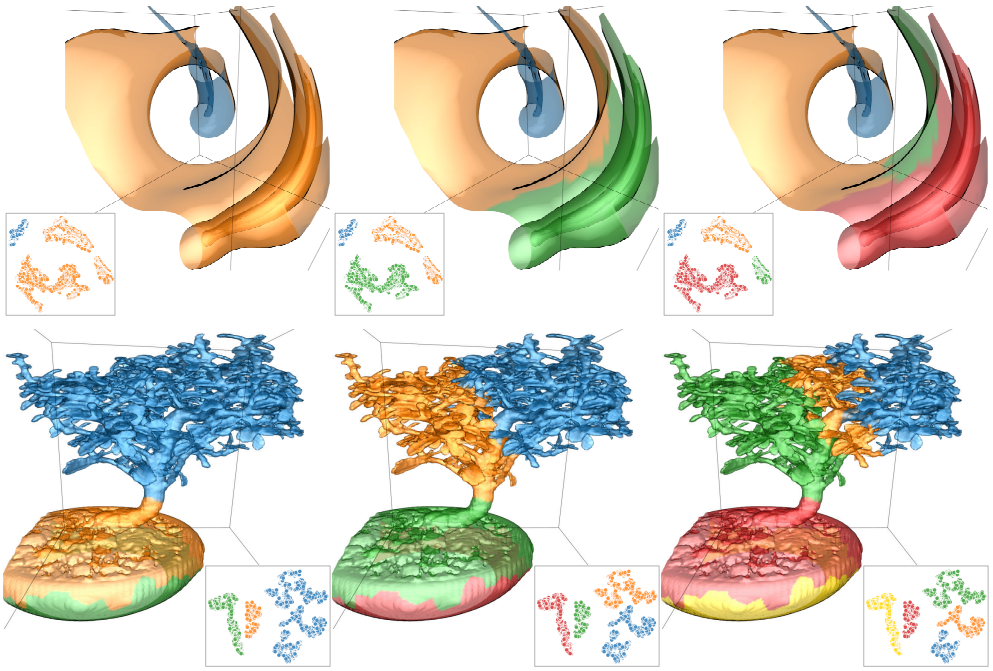
Figure 4.7: Evaluation of node features via brushing and linking. The unselected nodes are colored in gray.

Node clustering comparsion. In Figure 4.8, we qualitatively compare the node clustering results generated by GMMConv, EdgeConv, and SurfNet. The same number of clusters is used for the same data set. For the bonsai data set, GMMConv does not separate the bonsai from the basin, while both EdgeConv and SurfNet can correctly separate the two structures. For the lobster data set, both GMMConv and EdgeConv do not produce acceptable clustering results, while SurfNet can detect the three main structures (i.e., the claws, boy, and tail). The same conclusion can be drawn for the tornado data set, as only SurfNet partitions the tornado into three spirals of different curvature ranges. For the two swirls data set, GMMConv groups two spirals (refer to the green and orange parts) with



(a) GMMConv (b) EdgeConv (c) SurfNet

Figure 4.8: Node clustering results of a single surface. Top to bottom: bonsai, lobster, tornado, and two swirls. bonsai and lobster are isosufaces. tornado and two swirls are stream surfaces. Smaller surface clusters are highlighted with arrows of the same colors.



(a) (b) (c)

Figure 4.9: SurfNet node clustering results of a stream surface (top) using the five critical points data set and an isosurface (bottom) using the bonsai data set. From (a) to (c), the numbers of clusters for stream surfaces are 2, 3, and 4, and the numbers of clusters for isosurfaces are 3, 4, and 5.

overlap. EdgeConv does not detect the two spirals. SurfNet can recognize the surface's structure correctly, i.e., two spirals (refer to the blue and green parts) and one bridge (refer to the orange part). Overall, SurfNet outperforms GMMConv and EdgeConv by producing satisfactory node clustering results.

In Figure 4.9, we adjust the number of clusters to show the results with different scales (i.e., from coarse to fine) on a single stream surface and isosurface. All the results indicate that SurfNet can produce meaningful node clustering results under different numbers of clusters. For example, the clustering shows the basin's top and bottom portions and various parts of the bonsai. In Figure 4.10, we show more results with different data sets using SurfNet for stream surface clustering. Each column shows node clustering results for different surfaces of the same data set. These results further confirm the reliability of SurfNet in clustering the nodes of stream surfaces. We found that for complex flows (e.g.,

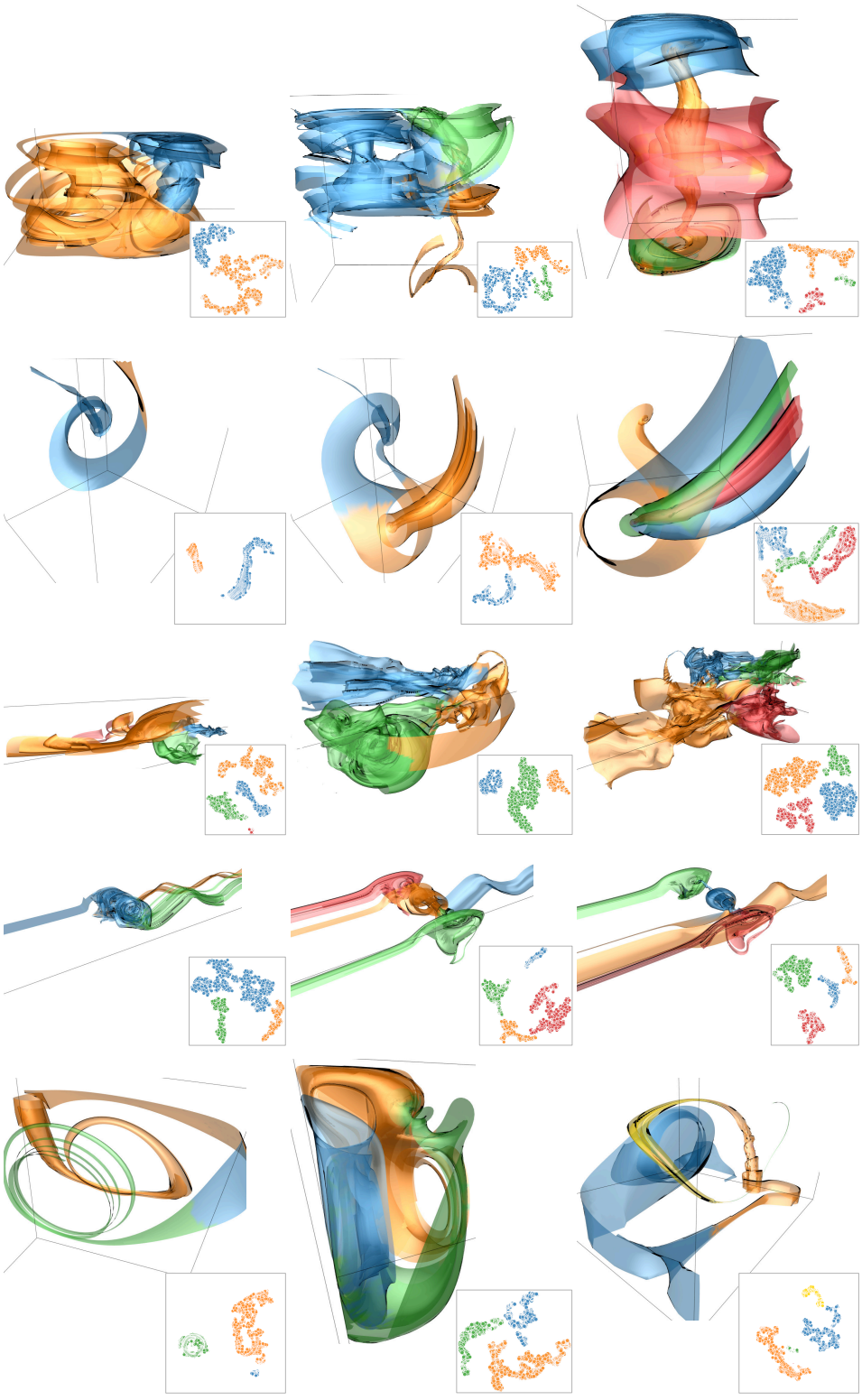
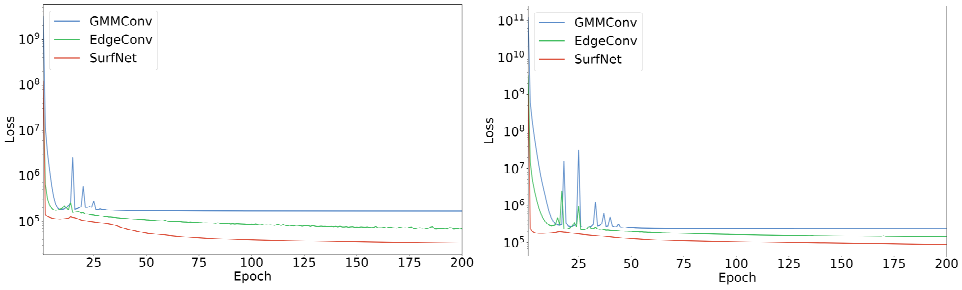
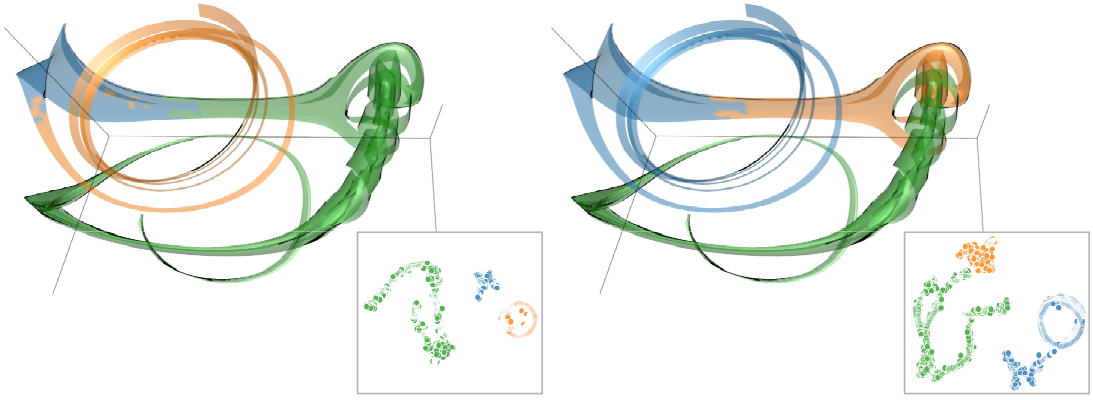


Figure 4.10: SurfNet node clustering results of a stream surface. Top to bottom: Bénard flow, five critical points, solar plume, square cylinder, and two swirls.



(a) five critical points (b) square cylinder

Figure 4.11: Loss convergence among GMMConv, EdgeConv, and SurfNet.



(a) joint training (b) separate training

Figure 4.12: SurfNet node clustering results of a stream surface using joint training and separate training. We utilize the tornado and two swirls data set to jointly train SurfNet.

solar plume), the clustering results generated by SurfNet include some minor errors, but the main surface structures can be detected correctly. In terms of training time, inference time, and model size, SurfNet, GMMConv, and EdgeConv do not exhibit significant differences, as shown in Table 4.4. The training time relies on the number of training samples and the complexity of surfaces (e.g., the number of nodes and edges on surfaces). As for the training time for isosurfaces, SurfNet only takes less than 10 seconds per epoch.

Baseline analysis. As shown in Figure 4.8, we observe that GMMConv does not produce acceptable node clustering results for all data sets, and EdgeConv only generates acceptable results for simple surfaces (e.g., bonsai). This is because GMMConv only ap-

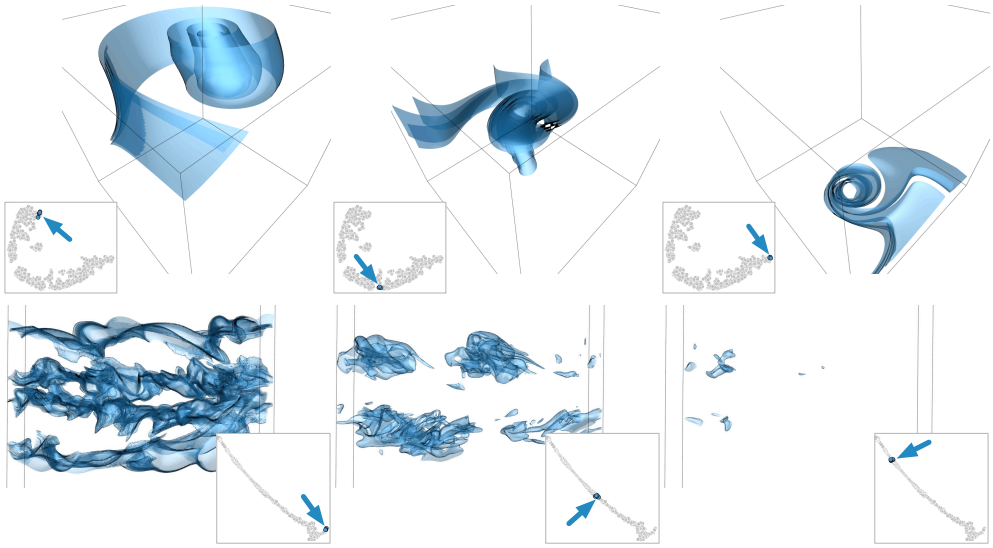
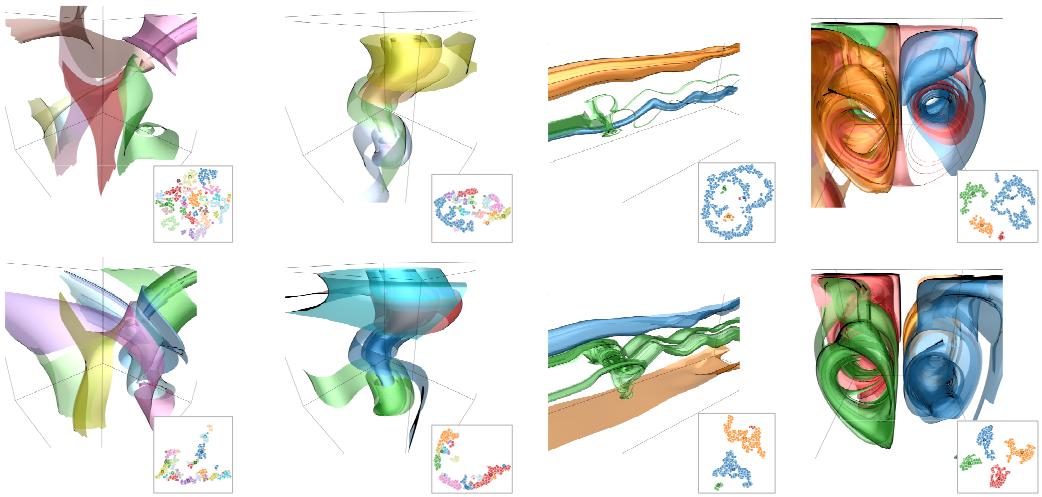


Figure 4.13: Evaluation of surface features using the tornado (top) and combustion (YOH) (bottom) data sets via brushing and linking. Surfaces corresponding to unselected points are not displayed.

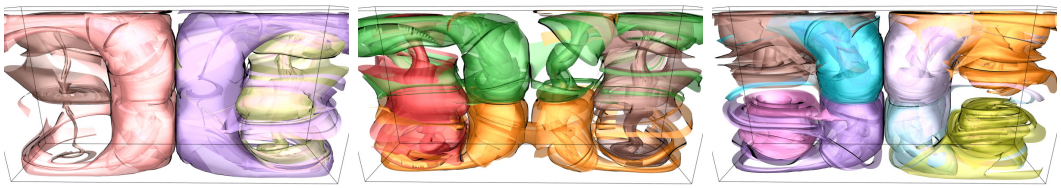
plies linear operations on the mesh Laplacian, which leads to slow convergence and poor performance. To verify this, in Figure 4.11, we plot the loss curves among GMMConv, EdgeConv, and SurfNet using the five critical points and square cylinder data sets. We can see that SurfNet and EdgeConv converge faster and exhibit more stable training compared with GMMConv. EdgeConv has limited capability to detect complex node patterns. It does not produce satisfactory clustering results for the lobster and two swirls data sets.

Performance degradation on boundary. As node clustering results show in Figures 4.8 and 4.10, the performance of node clustering degrades on the boundary of two structures. We give three explanations for such inaccurate boundary results. (1) *ambiguity*: The transition from one structure to another is a gradual change. This means that the shortest-path distance between two nodes located in two different patterns could be small, letting SurfNet treat the two nodes as similar. (2) *difficulty*: Detecting the boundary accurately in an unsupervised fashion is challenging since current deep learning frameworks require node annotations (e.g., experts label each node on the mesh) to delineate the bound-



(a) five critical points (b) tornado (c) square cylinder (d) two swirls

Figure 4.14: Representative stream surface selection results. Top row: FlowNet. Bottom row: SurfNet.

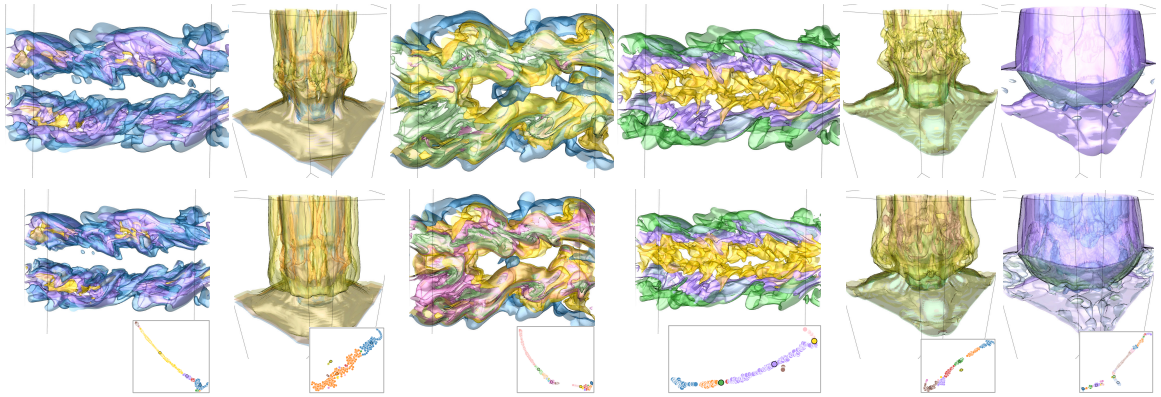


(a) (b) (c)

Figure 4.15: Customized SurfNet representative stream surface selection results using the Bénard flow data set. The numbers of representative surfaces are 4, 4, and 8, respectively, from (a) to (c).

ary [56, 158]. Even with these annotations, the boundary could still be inaccurate due to the uncertainty and error introduced by the annotations [185]. (3) *amplification*: The similarity of nodes located in two structures could be close. After several graph convolutions, the similarity could be amplified due to the weight-sharing mechanism and node propagation and aggregation in GCN, which hinders SurfNet from detecting a clear boundary between two structures.

Cross data set evaluation. To evaluate the cross data set generalization of SurfNet, we perform joint training using the tornado and two swirls data sets. Both joint training



(a) YOH (b) H (c) YOH → HR (d) YOH → MF (e) H → He (f) H → PD

Figure 4.16: Representative isosurface selection results. Top row: ISM. Bottom row: SurfNet. (a), (c), and (d): combustion. (b), (e), and (f): ionization. For SurfNet, (a) and (b) show same-variable inference results, while (c) to (f) show difference-variable inference results. The numbers of representative surfaces are 3, 3, 4, 3, 3, and 4, respectively, from (a) to (f).

and separate training take the same number of epochs for training. The surface clustering results are shown in Figure 4.12. For the two swirls data set, both ways of training can isolate the two swirls on the corresponding surfaces; however, separate training can also isolate the "bridge" connecting the two swirls. Therefore, we prefer using separate training.

4.2.3 Surface Selection

Baselines. For representative selection, we compare SurfNet against two surface selection methods:

- FlowNet [51] is a deep learning solution for representative stream surface selection. It encodes each surface into a latent feature and leverages t-SNE for dimensionality reduction and DBSCAN for surface clustering.
- Isosurface similarity maps (ISM) [16] is constructed by computing the mutual information between isosurfaces. The representative isosurfaces are selected based on ISM.

Note that we optimize SurfNet and FlowNet with the same number of samples (i.e., $1,000$ stream surfaces) and epochs for a fair comparison. Please refer to the accompanying videos for the frame-to-frame comparison of node clustering and representative selection results.

TABLE 4.4

TIME AND MODEL SIZE COMPARISON [1].

data set	method	training	inference	model size
five critical points	FlowNet	333	1.87	541
	GMMConv	19.18	0.005	0.83
	EdgeConv	14.84	0.006	0.84
	SurfNet	17.40	0.009	0.93
solar plume	FlowNet	1,001	2.42	1,228
	GMMConv	61.14	0.028	0.83
	EdgeConv	54.56	0.030	0.84
	SurfNet	50.85	0.034	0.93
square cylinder	FlowNet	255	1.83	791
	GMMConv	18.79	0.005	0.83
	EdgeConv	17.42	0.006	0.84
	SurfNet	18.36	0.009	0.93

Surface feature validation. To verify our surface embedding approach's effectiveness, we leverage t-SNE to project surface embeddings to a 2D space and then perform brushing and linking of these surfaces. The results are shown in Figure 4.13. The t-SNE projection conveys the surface's positional and shape information. For the tornado data set, the points on the left/right side of the projection view correspond to the surfaces shown at the top/bottom part of the surface view. For the combustion (YOH) data set, the points from right to left correspond to the isosurfaces with increasing isovalues. Thus, these brushing

[1] Average training time per epoch (in second) with 1,000 surface samples for training, average inference time (in second), and model size (MB). Note that FlowNet is designed for learning surface embedding, not node embedding.

and linking results demonstrate the meaningfulness of these surface features.

Representative selection. In Figure 4.14, we compare representative stream surface selection results between SurfNet and FlowNet [51]. Both methods allow users to handpick representative stream surfaces via a two-step process. Both methods pick the same number of representatives from the same set of stream surfaces for fair comparisons. We can see that compared to FlowNet, SurfNet chooses a subset of surfaces that better covers the domain to reveal more interesting flow features and patterns.

In Table 4.4, we report the average training time per epoch, average inference time, and the model size of FlowNet and SurfNet. SurfNet only needs 17 to 50 seconds to train $1,000$ samples per epoch, while FlowNet requires 200 to $1,000$ seconds. As for the inference time, SurfNet is still faster than FlowNet. Besides, SurfNet only needs 0.93 MB to store model parameters, while FlowNet needs 710 MB on average. This is because SurfNet is independent of the surface resolution, while FlowNet depends on the resolution of the vector field data. Another advantage of SurfNet is that it requires fewer training samples (i.e., $1,000$) for optimization since graph convolution operations are permutation invariant, and SurfNet utilizes both Euclidean and geodesic distances and a node's neighborhood information when producing embeddings. In contrast, FlowNet requires $2,000$ samples to train for most of the vector field data sets because convolutional operations are not permutation invariant, and FlowNet only considers neighborhood information when learning the feature representation.

In Figure 4.15, we can see that SurfNet can flexibly represent the underlying vector field with selected stream surfaces at different levels of detail. Users can determine the suitable number of representatives empirically as they adjust interactively.

In Figure 4.16 (a) and (b), we compare our representative isosurface selection results against those selected by ISM [16]. For fair comparisons, both methods select the same number of representatives from the same pool of isosurfaces. We can see that both methods can capture important isosurface features. Furthermore, in Figure 4.16 (c) to (f), we com-

pare the representative isosurface selection results through different-variable inference. In this case, we use variable YOH of the combustion data set for training and apply the network to HR and MF for inference. For the ionization data set, we use variable H for training and He and PD for inference. We can observe that these representative isosurfaces also exhibit comparable results as ISM.

4.2.4 Discussion

Although SurfNet produces coarser results in 3D shape compared with the existing solutions [134], SurfNet still has two advantages. (1) *Generalizability*: Once converged, SurfNet can group nodes on *unseen* surfaces, while traditional shape analysis approaches for meshes need to be rerun for producing new results. (2) *Diversity*: SurfNet can handle different kinds of surfaces while shape analysis algorithms are typically used to process *closed* meshes (e.g., 3D objects) but not *open* ones (e.g., stream surfaces). Also, SurfNet can handle node *clustering* and surface *selection*, while shape analysis only focuses on node-level rather than surface-level analysis. In addition, we believe it is a good start in unsupervised geometric learning for surface analysis and has great potential to improve in the future by incorporating domain knowledge into graph convolutional operations.

4.3 Conclusions

I have presented SurfNet, a new solution for clustering and selecting stream surfaces and isosurfaces. Based on the GCN, SurfNet can learn permutation-invariant node features from surfaces in an unsupervised manner, drawing a big difference from other works that learn node embeddings from labels and use different frameworks to solve surface clustering and selection separately. These node features encode their neighborhood information rather than purely physical information (e.g., position, normal). By projecting these embeddings into a 2D space, we provide an interactive visual interface for user exploration. Meaningful clustering and selection results are yielded under different clustering hyperpa-

rameters. These node embeddings produced from SurfNet preserve spatial proximity and geometric similarity.

I validate SurfNet on various vector and scalar field data sets of different patterns and compare the clustering and selection results derived from the learned node embeddings with those produced using other state-of-the-art methods. Compared with FlowNet, training SurfNet is 10 to 20 times faster per epoch and inferring is 70 to 170 times faster while the model's storage-saving is 300 to $1,300$ folds (Table 4.4). Qualitative results show that SurfNet generates better node clustering results than those generated by GMMConv and EdgeConv. Qualitative and quantitative results show that SurfNet suggests comparable or better representative selection results than those generated by FlowNet (stream surfaces) and ISM (isosurfaces).

CHAPTER 5

V2V: A DEEP LEARNING APPROACH TO VARIABLE-TO-VARIABLE SELECTION AND TRANSLATION FOR MULTIVARIATE TIME-VARYING DATA

5.1 V2V

5.1.1 Overview

Let us denote $\mathbf{V}^{\text{var}} = \{\mathbf{V}^{\text{var}_1}, \mathbf{V}^{\text{var}_2}, \cdots, \mathbf{V}^{\text{var}_m}\}$ as a set of variables in the given MTVD, and $\mathbf{V}^{\text{var}_i} = \{\mathbf{V}_1^{\text{var}_i}, \mathbf{V}_2^{\text{var}_i}, \cdots, \mathbf{V}_n^{\text{var}_i}\}$ as the temporal sequence of variable i, where m is the number of variables and n is the number of time steps. $\mathbf{V}^{\text{var}_i}[1:k]$ is a subset of $\mathbf{V}^{\text{var}_i}$, which has the first k time steps ($n \gg k$). $\mathbf{V}^{\text{var}_i}[1:k]$ is also the samples I take to train our deep learning model. $\mathbf{V}^T$ and $\mathbf{V}^S$ denote, respectively, the ground truth (GT) and synthesized variables from V2V. $\mathbf{F}_j^{\text{var}_i}$ is the feature of variable i at the jth time step. Finally, let L, H, and W be the spatial dimensions of $\mathbf{V}^{\text{var}}$.

Our goal is to learn a mapping function $\mathscr{T}$ from one variable sequence $\mathbf{V}^{\text{var}_a}$ to another variable sequence $\mathbf{V}^{\text{var}_b}$, namely, $\mathbf{V}^{\text{var}_b} = \mathscr{T}(\mathbf{V}^{\text{var}_a})$. As sketched in Figure 5.1 (a), our approach consists of three stages: *feature learning*, *translation graph construction*, and *variable translation*. At the feature learning stage, I collect the available time steps from all variables of the given MTVD and utilize a U-Net [126] to learn their latent features. Then I leverage t-SNE [151] for dimensionality reduction where the latent feature of per variable and per time step is projected onto a 2D space. The t-SNE projection helps us analyze and understand the similarities and differences among these variables, which provide us hints on whether or not a given pair of variables is transferable. Among all the variables, I select a transferable variable group (e.g., $\{\mathbf{V}^{\text{var}_1}, \cdots, \mathbf{V}^{\text{var}_p}\}$), where $p \leq m$. As an example, the

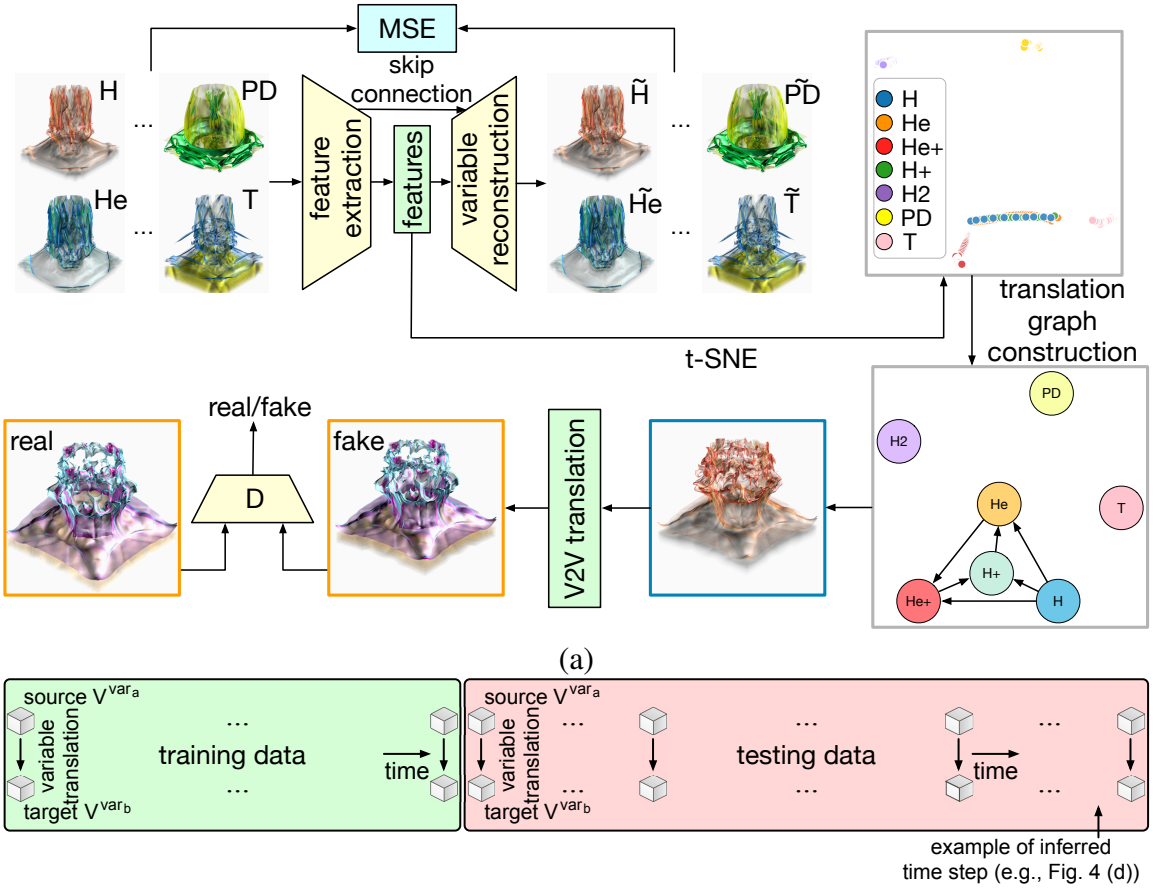


Figure 5.1: (a) Overview of V2V. For feature learning, a U-Net is applied to extract features from variables and t-SNE is used to project the features for estimating variable similarity. A translation graph is constructed based on the learned variable features. For variable translation, variable pairs are selected and V2V is trained for learning the translation mapping. (b) Training and testing data from the volume sequence.

transferable variable group for the example shown in Figure 5.1 is {H, H+, He, He+}.

At the translation graph construction stage, given the transferable variable group, I estimate the transferable difficulty of variable b conditioned on variable a, and construct a translation graph $\mathscr{G}$ based on the computed transferable difficulty among different variables. Then, the source variable and target variable are selected from $\mathscr{G}$.

At the variable translation stage, I train a V2V network to learn the mapping between the two variable sequences (i.e., $\mathbf{V}^{\text{var}_a} \rightarrow \mathbf{V}^{\text{var}_b}$) based on the translation graph result. Our V2V includes one generator (G) and one discriminator (D). G consists of three modules: *feature extraction*, *feature translation*, and *variable translation*. The feature extraction module extracts rich semantic information from the input variable. The feature translation module translates the features from the source variable to the target variable at different scales. The variable translation module translates the refined features to the target variable domain. As shown in Figure 5.1 (b), in our experiments, the training data consist of early time steps of $\mathbf{V}^{\text{var}_a}$ and $\mathbf{V}^{\text{var}_b}$, and the testing data consist of later time steps of $\mathbf{V}^{\text{var}_a}$ and $\mathbf{V}^{\text{var}_b}$.

For U-Net training, I utilize the mean squared error (MSE) as the loss function to compute the difference between the reconstructed and GT variables. For V2V training, I leverage *adversarial*, *volumetric*, and *feature* losses to optimize the network. Next, I describe our approach in detail, including the network architectures of feature learning and variable translation, as well as the algorithm for constructing the translation graph.

5.1.2 Feature Learning

At the feature learning stage, I leverage a U-Net that takes the available time steps of all variables from the same MTVD as input and outputs feature descriptors per variable and per time step. U-Net also allows the reconstruction of a variable at a time step from the corresponding feature descriptor. Skip connections in U-Net can bridge different semantic features of the same scale, avoiding information loss during reconstruction.

In general, U-Net is composed of an *encoding path* and a *decoding path*. There are four convolutional (Conv) layers in the encoding path and four composites of deconvolutional (DeConv) layers and Conv layers in the decoding path. In the encoding path, each of the first three Conv layers reduces the input's dimension by half. The feature maps start with 64 and double in the following Conv layers. Then, I apply one Conv layer to transform the learned features into a 1D vector with 512 components. In the decoding path, DeConv layers are utilized to upscale the feature back to the original dimension, and the following Conv layers are utilized to fuse and refine feature maps. To keep the information flowing smoothly and avoid information loss, I concatenate the feature maps from the Conv layer in the encoding path and the feature maps from the DeConv layer as input for the consecutive Conv layer for refinement. The feature maps start with 256 and reduce by half in the following DeConv layers. I keep the same feature maps in each Conv layer followed by each DeConv layer. Note that the concatenation happens at the corresponding scale (i.e., these two tensors have the same resolution). Rectified linear unit (ReLU) [107] is utilized after each Conv or DeConv layer to help the network learn faster and perform better. After the final Conv layer, $\tanh(\cdot)$ is applied for normalization (in the range of $[-1,1]$).

Loss function. In order to ensure that the synthesized variables are close to the GT variables, I use MSE as the loss function to train U-Net. The MSE loss is defined as

$$\mathscr{L} = \sum_{j=1}^{u} ||\hat{\mathbf{V}}_j - \mathbf{V}_j||_2, \tag{5.1}$$

where $\mathbf{V}_j$ and $\hat{\mathbf{V}}_j$ are, respectively, the GT and synthesized variables of the jth training sample, $u = k \times m$ is the number of training samples, and $||\cdot||_2$ is L_2 norm.

5.1.3 Translation Graph Construction

After the transferable variable group (e.g., $\{\mathbf{V}^{\text{var}_1}, \cdots, \mathbf{V}^{\text{var}_p}\}$) is determined, I define

Algorithm 1: Translation graph construction.

Require: A set of variables: $\{\mathbf{V}^{\text{var}_1}, \mathbf{V}^{\text{var}_2}, \cdots, \mathbf{V}^{\text{var}_p}\}$.
  initialize a translation graph $\mathscr{G}$ with $\text{var}_1, \cdots, \text{var}_p$ as nodes and no edge
  for $i = 1 \cdots p$ **do**
    for $j = 1 \cdots i$ **do**
      if $\mathscr{E}(\mathbf{F}^{\text{var}_i}, \mathbf{F}^{\text{var}_j}) < \varepsilon$ **then**
        Compute $\text{TD}(\mathbf{V}^{\text{var}_j} || \mathbf{V}^{\text{var}_i})$ and $\text{TD}(\mathbf{V}^{\text{var}_i} || \mathbf{V}^{\text{var}_j})$
        if $\text{TD}(\mathbf{V}^{\text{var}_j} || \mathbf{V}^{\text{var}_i}) < \text{TD}(\mathbf{V}^{\text{var}_i} || \mathbf{V}^{\text{var}_j})$ **then**
          add an edge from var_i to var_j to $\mathscr{G}$
        else
          add an edge from var_j to var_i to $\mathscr{G}$
        end if
      end if
    end for
  end for
  return $\mathscr{G}$

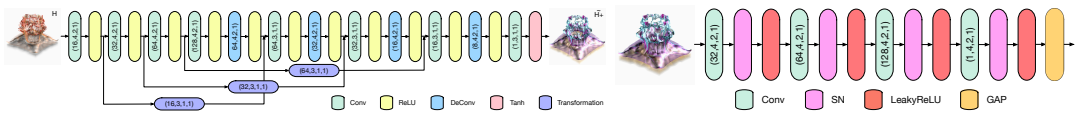
the *transferable difficulty* (TD) of $\mathbf{V}^{\text{var}_j}$ conditioned on $\mathbf{V}^{\text{var}_i}$ as follows

$$\text{TD}(\mathbf{V}^{\text{var}_j} || \mathbf{V}^{\text{var}_i}) = \frac{1}{k} \sum_{t=1}^{k} \text{KL}(\mathbf{F}_t^{\text{var}_j} || \mathbf{F}_t^{\text{var}_i}), \tag{5.2}$$

where $\mathbf{F}_t^{\text{var}_j}$ is the feature of variable j at time step t, $\text{KL}(\cdot||\cdot)$ is Kullback-Leibler divergence, and k is the total available time steps. The transferable order for the variable pair i and j is given by

$$\min\{\text{TD}(\mathbf{V}^{\text{var}_j} || \mathbf{V}^{\text{var}_i}), \text{TD}(\mathbf{V}^{\text{var}_i} || \mathbf{V}^{\text{var}_j})\}. \tag{5.3}$$

A translation graph $\mathscr{G}$ can be constructed based on the calculation of TDs between different variable pairs. The process is described in Algorithm 1. Given a pair of variables, I first compute the Euclidean distance of these two variables in the feature space. The distance determines whether or not these two variables are transferable. If the distance is less than a threshold ε, then I compute TD for the two variables, and determine the transferable order based on Equation 5.3.



(a) Generator G (b) Discriminator D

Figure 5.2: Network architecture of V2V. (a) G contains eight Conv layers, four DeConv layers, and three transformation blocks. (b) D includes four Conv layers, four SN layers, and one GAP layer.

5.1.4 Variable Translation

Generator. G consists of three modules. The feature extraction module utilizes four Conv layers to extract the features from the input variable. Each Conv downscales the input by half, and a ReLU is followed to accelerate the training and improve model performance. The feature translation module leverages three *transformation blocks* to translate the features from the source variable to the target variable at different scales and feed into the variable translation module. Each transformation block includes two paths. One path contains three Conv layers, and the other path contains one Conv layer. Finally, these two paths are connected by addition. The variable translation module utilizes four DeConv layers and four Conv layers followed by ReLU to map the feature to the output variable domain. Each DeConv upscales the input twice. In addition, after each DeConv layer, I stack the outputs from DeConv and the feature translation stage together and feed into a Conv layer. The two stacked outputs share the same scale, and the Conv layer does not change the scale of the input. Note that I only use $\tanh(\cdot)$ in the final Conv layer. The architecture of G is sketched in Figure 5.2 (a).

Discriminator. D includes four Conv layers, four spectral normalization (SN) [104] layers, and one global average pooling (GAP) [90] layer. Each Conv downscales the input by half, and SN is followed by Conv to normalize the weights in Conv for training stabilization. Leaky ReLU activation ($\alpha = 0.2$) is applied after each Conv layer. Finally, one GAP is leveraged to squeeze the output into a tensor with $1 \times 1 \times 1 \times 1$. No activation function is added after GAP. The architecture of D is sketched in Figure 5.2 (b).

Loss function. As suggested by Han and Wang [46], I apply adversarial, volumetric, and feature losses to optimize V2V so that the synthesized variables are close to the GT variables. The adversarial loss is defined as

$$\min_{\theta_G} \mathscr{L}_G = \mathbb{E}_{V \in \mathbf{V}^S}\big[(D(G(V)) - 1)^2\big], \tag{5.4}$$

$$\min_{\theta_D} \mathscr{L}_D = \frac{1}{2}\mathbb{E}_{V \in \mathbf{V}^S}\big[D(V)\big] + \frac{1}{2}\mathbb{E}_{V \in \mathbf{V}^T}\big[(D(G(V)) - 1)^2\big], \tag{5.5}$$

where θ_G and θ_D are the learnable parameters in G and D, and $\mathbb{E}[\cdot]$ denotes the expectation operation.

The volumetric loss is defined as

$$\mathscr{L}_V = \mathbb{E}_{V' \in \mathbf{V}^S, V \in \mathbf{V}^T}\big[||G(V') - V||_2\big], \tag{5.6}$$

where $||\cdot||_2$ denotes the L_2 norm.

The feature loss is defined as

$$\mathscr{L}_F = \mathbb{E}_{V' \in \mathbf{V}^S, V \in \mathbf{V}^T} \sum_{k=1}^{N} \frac{1}{N_k}\big[||F^k(G(V')) - F^k(V)||_1\big], \tag{5.7}$$

where N is the total number of Conv layers in D, N_k is the number of elements in the kth Conv layer, and $F^k(\cdot)$ denotes the feature representation at the kth Conv layer.

The overall loss for G is the combination of the three losses

$$\min_{\theta_G} \mathscr{L}_G = \lambda_1\big(\mathbb{E}_{V \in \mathbf{V}^T}\big[\big(D(G(V)) - 1\big)^2\big]\big) + \lambda_2 \mathscr{L}_V + \lambda_3 \mathscr{L}_F, \tag{5.8}$$

where λ_1, λ_2, and λ_3 are weights, each in the range of $[0, 1]$.

Note that adversarial, volumetric, and feature losses serve different purposes in V2V training. Adversarial loss aims to judge the realness of the synthesized volumes from the generator. Volumetric loss seeks to ensure that the synthesized volumes are close to the GT

volumes. Feature loss aims to stabilize the training process and enhance the visual quality.

TABLE 5.1

THE VARIABLES AND DIMENSIONS OF EACH DATA SET

data set	variables	dimension ($x \times y \times z \times t$)
climate	SALT, TEMP	$360 \times 66 \times 27 \times 200$
combustion	CHI, HR, MF, YOH	$480 \times 720 \times 120 \times 100$
ionization	H, H+, He, He+, H2, PD, T	$600 \times 248 \times 248 \times 100$

5.2 Results and Discussion

5.2.1 Data Sets and Network Training

I experimented with our approach using the data sets listed in Table 5.1. I implemented V2V based on PyTorch [113] and used a single NVIDIA TESLA P100 GPU for training. For feature learning, I used the bicubic kernel with a downscaling factor of four to downscale combustion and ionization data sets for fast training. For variable translation, I used the original resolution for training; however, for each epoch, I randomly crop the volumes. This cropping mechanism can reduce training cost and GPU memory consumption. I point out that V2V can be applied to volumes of arbitrary size because it is fully convolutional. I scaled the range of $\mathbf{V}^{\text{var}}$ to $[-1,1]$. All learnable parameters in U-Net and V2V are initialized using He et al. [57] and the Adam algorithm [78] is applied for parameter update. I set one training sample per mini-batch. For training U-Net, the learning rate is set to 10^{-4}. For training V2V, different learning rates for G and D are set as suggested by Roth et al. [128].

The learning rates for G and D are 10^{-4} and 4×10^{-4}, respectively. $\beta_1 = 0.0$, $\beta_2 = 0.999$. $\lambda_1 = 10^{-3}$, $\lambda_2 = 1$, and $\lambda_3 = 5 \times 10^{-1}$. I trained U-Net and V2V for 50 and 150 epochs for all data sets, respectively. I sampled the first 40% time steps for training and the rest for inference. All these hyperparameters are determined based on experiments.

5.2.2 Results

Baselines. I compare one baseline solution for variable selection:

- Biswas et al. [13]: It is an information-theoretic approach for variable grouping. Once grouped, users can select representative variables for further exploration. I leverage this solution to select variable pairs as the input to the V2V translation task.

Note that Biswas et al. is a solution for variable selection *only*, and *not* for variable translation. For translation comparison, I implement three baseline solutions for the V2V translation task:

- Histogram matching (HM) [129]: HM is a traditional approach for translating one data set to another one conditioned on the content and style of the data. I apply HM to translate variable j at time step k conditioned on variable i at time step k and variable j at time step $k-1$.
- Pix2Pix [69]: Pix2Pix is the first *paired* image-to-image translation framework. The original Pix2Pix architecture is leveraged for the V2V translation task.
- CycleGAN [187]: CycleGAN is a deep learning solution for *unpaired* image-to-image translation. Since the variables are paired in our V2V translation task, I replace the *identity* loss in CycleGAN with the *volumetric* loss in V2V.

For a fair comparison, I use the same loss functions (i.e., adversarial, volumetric, and feature losses) designed for V2V to train Pix2Pix and CycleGAN.

Unless otherwise stated, I display all visualization results using the inferred volumes (refer to Figure 5.1 (b) for an example). The same settings for lighting, viewing, transfer function (for direct volume rendering), and isovalue (for isosurface rendering) are applied

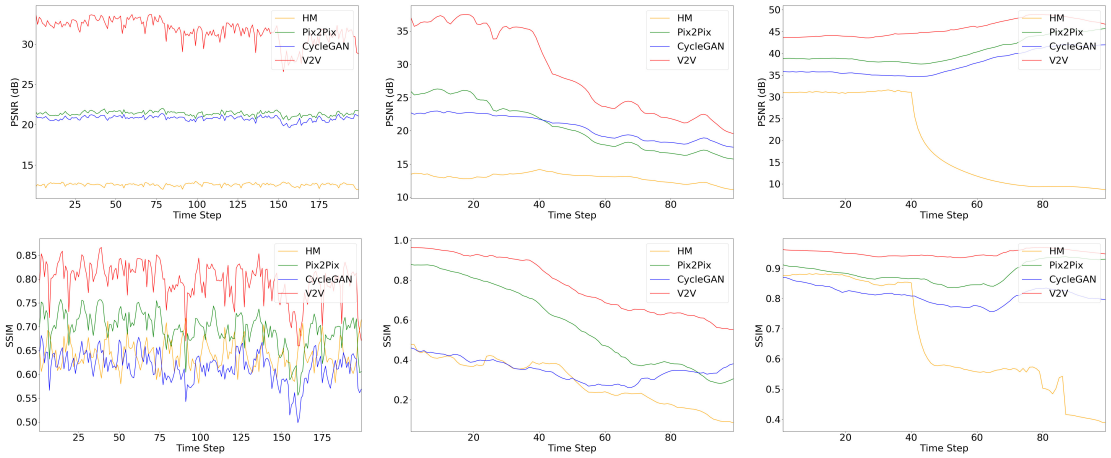
to all visualization results for the same data set. With respect to the GT, I compare our V2V results with those of HM, Pix2Pix, and CycleGAN.

Evaluation metrics. For quantitative evaluation, I compute, between the synthesized variables and GT variables, the *peak signal-to-noise* (PSNR) at the data level, *structural similarity index* (SSIM) at the image level, and *isosurface similarity* (IS) [46] at the feature level.

TABLE 5.2

AVERAGE PSNR AND SSIM VALUES

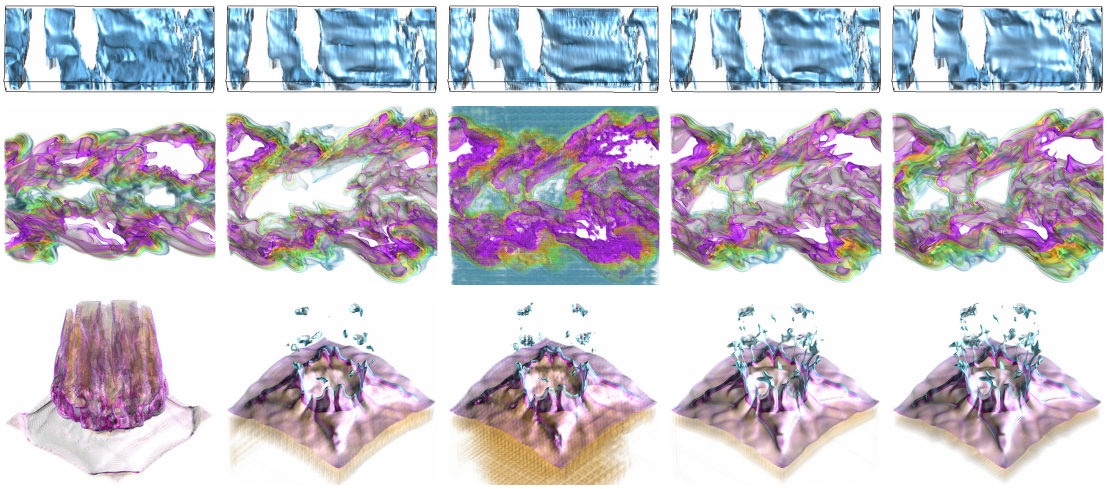
data set ($v_1 \rightarrow v_2$)	method	PSNR (dB)	SSIM
climate (SALT→TEMP)	HM	13.12	0.642
	Pix2Pix	21.39	0.695
	CycleGAN	20.78	0.616
	V2V	**31.69**	**0.797**
combustion (MF→YOH)	HM	12.96	0.291
	Pix2Pix	20.46	0.585
	CycleGAN	20.56	0.351
	V2V	**28.73**	**0.776**
ionization (H→H+)	HM	19.62	0.668
	Pix2Pix	40.58	0.887
	CycleGAN	37.59	0.812
	V2V	**45.75**	**0.951**



(a) climate (SALT→TEMP) (b) combustion (MF→YOH) (c) ionization (H→H+)

Figure 5.3: PSNR (top row) and SSIM (bottom row) of synthesized variables (TEMP, YOH, and H+) under HM, Pix2Pix, CycleGAN, and V2V.

Quantitative and qualitative analysis. In Figure 5.3, I show the data (PSNR) and image (SSIM) level results using HM, Pix2Pix, CycleGAN, and V2V. At the data level, for the climate (SALT→TEMP) data set, all four curves exhibit a periodic pattern since each time step denotes the temperature for each month and 12 time steps are for one year. The PSNR values of V2V outperform those of HM, Pix2Pix, and CycleGAN. For the combustion (MF→YOH) data set, PSNR values decrease as time step goes. This is because, at the later time steps, the temporal behavior becomes more turbulent and complex, making the prediction more difficult. Again, V2V still outperforms HM, Pix2Pix, and CycleGAN. For the ionization (H→H+) data set, it is clear that V2V achieves the highest PSNR values for each time step. At the image level, V2V can still produce higher SSIM values compared with HM, Pix2Pix, and CycleGAN. It is the clear winner for the climate (SALT→TEMP), combustion (MF→YOH), and ionization (H→H+) data sets. For the combustion (MF→YOH) data set, due to the increase of visual content, the SSIM values decrease as time step goes. In Table 5.2, the average PSNR and SSIM values for HM, Pix2Pix, CycleGAN, and V2V are reported. Again, V2V achieves the best PSNR and SSIM values. Note that the PSNR and SSIM curves of HM suddenly decrease after time



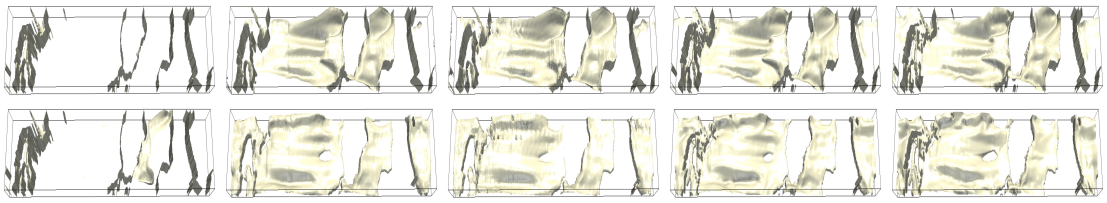
(a) HM (b) Pix2Pix (c) CycleGAN (d) V2V (e) GT

Figure 5.4: Comparison of volume rendering results. Top to bottom: the climate (SALT→TEMP), combustion (MF→YOH), and ionization (H→H+) data sets. Displayed here are the renderings of TEMP at time step 159, YOH at time step 65, and H+ at time step 70, respectively.

step 40 for the combustion and ionization data sets since I only use 40% data for training. The error accumulates when predicting the later time steps. Since the climate data set is periodic, the PSNR and SSIM curves of HM do not exhibit a similar pattern as that of the other two data sets.

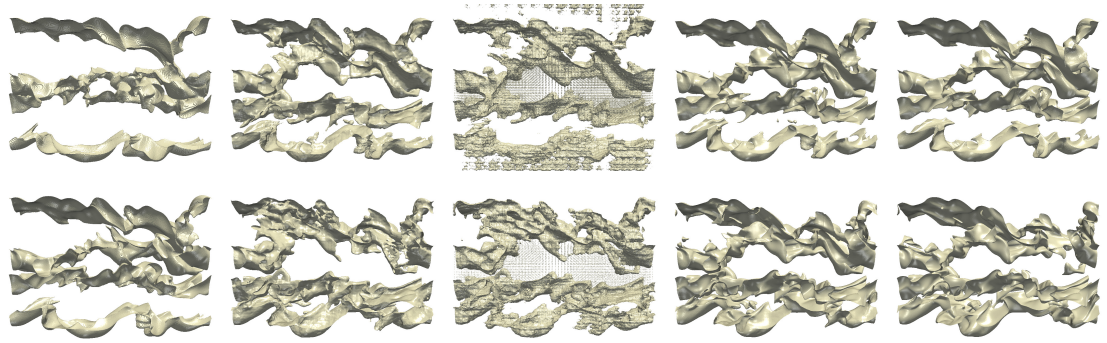
In Figure 5.4, the volume rendering results of the volumes synthesized by HM, Pix2Pix, CycleGAN, and V2V are shown. For the climate (SALT→TEMP) data set, the rendering results synthesized by Pix2Pix and CycleGAN contain artifacts. The result generated by HM cannot well capture the main structure, while the result produced by V2V is much smoother and similar to the GT. For the combustion (MF→HR) data set, V2V produces finer details with respect to GT, while HM and CycleGAN fail to recover the volume well. Pix2Pix generates some artifacts and is unable to recover the content around the volume boundary. For the ionization (H→H+) data set, V2V achieves the best result compared with HM, Pix2Pix, and CycleGAN. For example, for the Pix2Pix result, there are fewer details at the top part, and there are some artifacts at the bottom layer. For the CycleGAN result, it produces more orange content at the bottom part and fails to accurately recover

the top part. For the HM result, it generates more purple and yellow content at the top part.



(a) HM (b) Pix2Pix (c) CycleGAN (d) V2V (e) GT

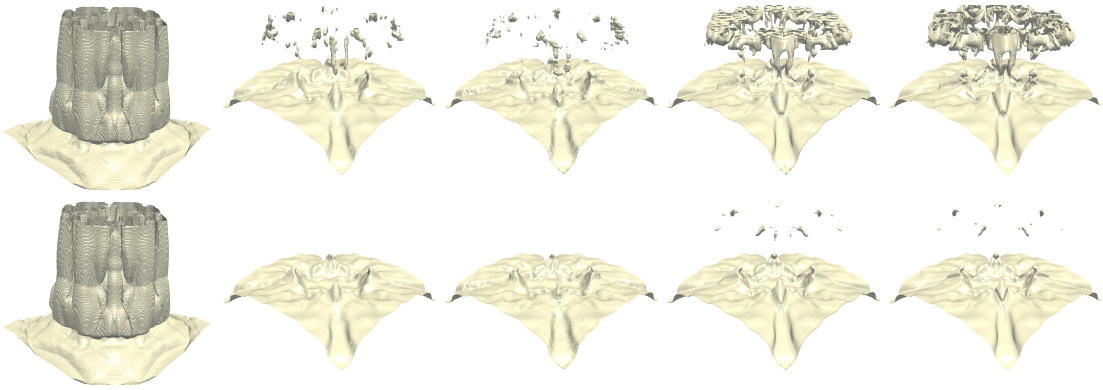
Figure 5.5: Comparison of isosurface rendering results of the climate (SALT→TEMP) data set at time step 167. The chosen isovalues are $v = -0.4$ (top row) and $v = 0.3$ (bottom row).



(a) HM (b) Pix2Pix (c) CycleGAN (d) V2V (e) GT

Figure 5.6: Comparison of isosurface rendering results of the combustion (MF→YOH) data set at time step 53. The chosen isovalues are $v = -0.9$ (top row) and $v = -0.55$ (bottom row).

In Figures 5.5, 5.6, and 5.7, the isosurface rendering results of the volumes synthesized by HM, Pix2Pix, CycleGAN, and V2V using the climate (SALT→TEMP), combustion (MF→YOH), and ionization (H→H+) data sets are displayed. For each data set, I choose



(a) HM (b) Pix2Pix (c) CycleGAN (d) V2V (e) GT

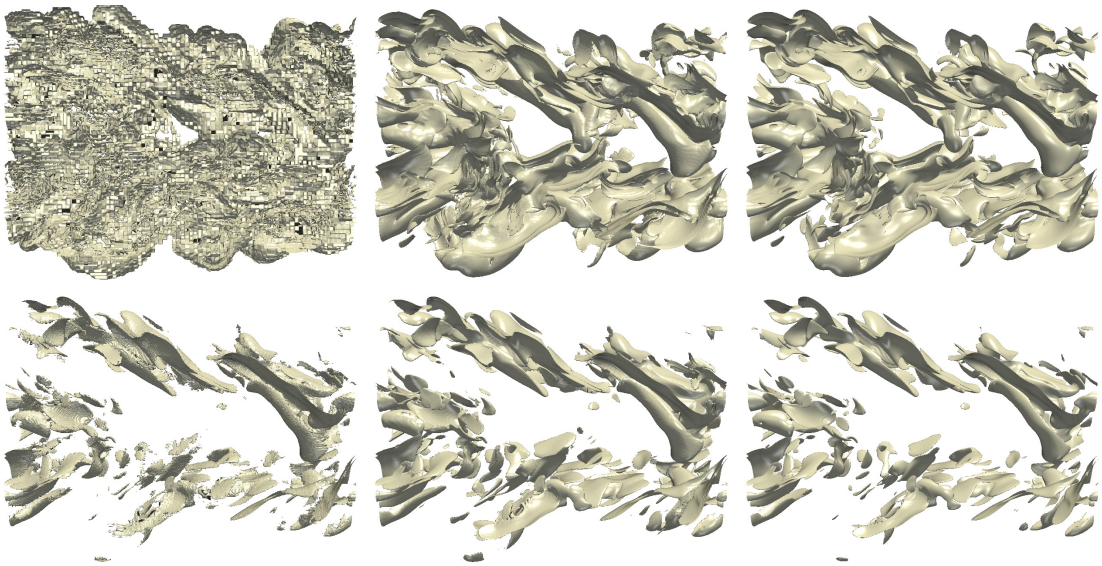
Figure 5.7: Comparison of isosurface rendering results of the ionization (H→H+) data set at time step 92. The chosen isovalues are $v = -0.96$ (top row) and $v = -0.9$ (bottom row).

one time step and two isovalues to render the isosurfaces. For the climate (SALT→TEMP) data set, it is evident that V2V can generate the highest quality isosurfaces compared with HM, Pix2Pix, and CycleGAN. HM fails to construct the isosurfaces close to GT, and the isosurfaces extracted from Pix2Pix and CycleGAN are filled with noises and artifacts. Similar observations can be made for the combustion (MF→YOH) data set where V2V generates the highest quality isosurfaces compared with HM, Pix2Pix, and CycleGAN. For the ionization (H→H+) data set, V2V produces the highest quality isosurfaces. Pix2Pix and CycleGAN fail to construct the isosurface at the top part, and HM synthesizes fake features compared with the GT results. Furthermore, the average IS values for these three data sets are reported in Table 5.3. The average IS values also demonstrate that V2V achieves the best quality. Moreover, among Pix2Pix, CycleGAN, and V2V, CycleGAN has almost the worst performance in terms of PSNR, SSIM, and IS. This is because, unlike image-to-image translation, where the translation is symmetric (e.g., day to night), in V2V translation, the translation is asymmetric (e.g., it is more challenging to translate from CHI to MF compared with translating from MF to CHI). Therefore, adding cycle consistency will hurt the translation performance. As for Pix2Pix, this architecture is too simple to capture the complex structure changes between the variables.

TABLE 5.3

AVERAGE IS VALUES AT CHOSEN ISOVALUES

	HM		Pix2Pix		CycleGAN		V2V	
data set ($v_1 \rightarrow v_2$)	$v = -0.4$	$v = 0.3$	$v = -0.4$	$v = 0.3$	$v = -0.4$	$v = 0.3$	$v = -0.4$	$v = 0.3$
climate (SALT→TEMP)	0.12	0.15	0.83	0.85	0.73	0.79	**0.92**	**0.91**
	$v = -0.9$	$v = 0.65$	$v = -0.55$	$v = -0.9$	$v = -0.55$	$v = -0.9$	$v = -0.9$	$v = -0.55$
combustion (MF→YOH)	0.23	0.19	0.72	0.69	0.47	0.49	**0.82**	**0.84**
	$v = -0.96$	$v = -0.9$	$v = -0.96$	$v = -0.9$	$v = -0.96$	$v = -0.9$	$v = -0.96$	$v = 0.9$
ionization (H→H+)	0.32	0.41	0.82	0.84	0.79	0.81	**0.92**	**0.95**



(a) LC [86, 87] (b) V2V (c) GT

Figure 5.8: Isosurface rendering results using the combustion (CHI) data set at time step 60. The chosen isovalues are $v = -0.7$ (top row) and $v = 0.3$ (bottom row).

In Table 5.4, I report the total training time (in hour), the average inference time (in second), and model size for Pix2Pix, CycleGAN, and V2V, respectively. As I can see, CycleGAN takes the longest training time since it needs to go through the network six times in one iteration (i.e., two discriminators, one cycle of X→Y→X, and another cycle of Y→X→Y). Pix2Pix and V2V only need to go through the network twice in one iteration (i.e., one discriminator and one generator). As for the inference time, there is no significant difference. In terms of model size, V2V needs 14MB to store parameters.

Comparison against compression. In Figure 5.8, I compare V2V and an advanced lossy compression (LC) method [86, 87] using isosurface rendering results. This LC method can achieve a high compression rate while producing less data distortion. To achieve a fair comparison, I set a similar PSNR value (i.e., 29.5 dB) for both approaches. As I can see, the isosurfaces generated by LC include significant noises and artifacts compared with those produced by V2V.

Evaluation of variable selection.

TABLE 5.4

TIME AND MODEL SIZE COMPARISON [1]

data set	method	training epochs	training time	inference time	model size
climate	Pix2Pix	150	6.79	3.52	6
	CycleGAN	200	56.44	4.63	26
	V2V	150	15.36	4.01	14
combustion	Pix2Pix	150	31.21	187.45	6
	CycleGAN	200	169.08	220.36	26
	V2V	150	53.14	194.72	14
ionization	Pix2Pix	150	21.43	122.39	6
	CycleGAN	200	125.46	130.93	26
	V2V	150	40.76	129.07	14

To show the effectiveness of the proposed variable selection solution, I compare V2V against Biswas et al. [13]. I only use Biswas et al. to choose transferable variable pairs, as it does not perform variable translation. Once the pairs are selected from either V2V or Biswas et al. I apply the same solution (i.e., V2V) for translation. The training time of the variable selection stage for the combustion and ionization data sets is 1.86 and 2.14 hours, respectively. The training time depends on the number of variables and the dimension of the data set. In Figure 5.9, I show clustered graphs and translation graphs of the combustion and ionization data sets using Biswas et al. and V2V, respectively. Note that the clustered graphs of Biswas et al. are fully-connected and undirected, while the translation graphs of V2V are partially-connected and directed. For the combustion data set, Biswas et al.

[1] Total training time (in hour), average inference time (in second), and model size (MB) under Pix2Pix, CycleGAN, and V2V.

TABLE 5.5

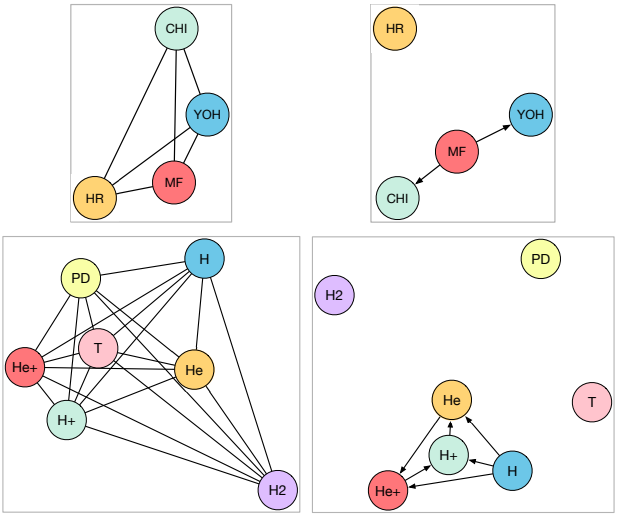
PERFORMANCE COMPARISON FOR VARIABLE TRANSLATIONS USING BISWAS ET AL. [13] AND V2V

data set	variable pair	approach	PSNR (dB)	SSIM
combustion	YOH→CHI	Biswas et al.	24.76	0.607
	MF→CHI	V2V	**35.76**	**0.829**
ionization	T→H+	Biswas et al.	33.41	0.827
	H→H+	V2V	**45.75**	**0.951**

demonstrates that YOH and CHI are more similar compared with MF and CHI, while V2V leads to the opposite conclusion. For the ionization data set, Biswas et al. demonstrates that T and H+ are similar, while H and H+ are distinguishable; however, V2V gets the opposite results.

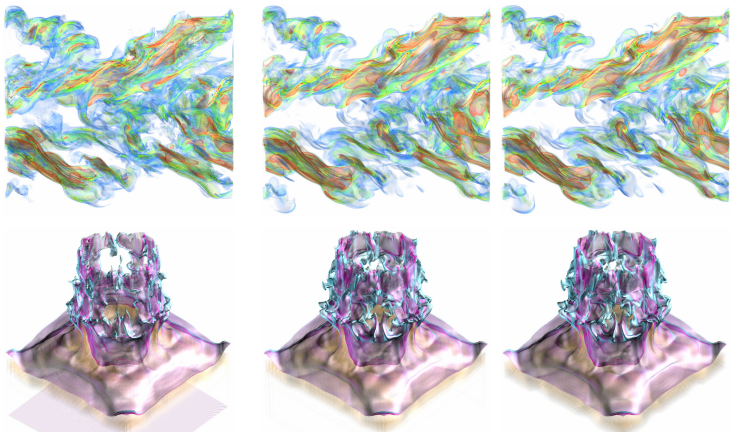
To evaluate the effectiveness of these two variable selection approaches, I choose two pairs from Biswas et al. (i.e., YOH→CHI and T→H+) and two from V2V (i.e., MF→CHI and H→H+) for the translation task. The results are shown in Figures 5.10 and 5.11. As I can see, for Biswas et al. YOH→CHI and T→H+ are not successfully judged from both volume and isosurface rendering results. For example, the volume rendering of CHI and H+ cannot exhibit a good visual quality compared with GT. As for the variable pairs selected by V2V, the translation results are satisfactory. Table 5.5 reports the average PSNR and SSIM values under these two variable translation schemes. Overall, based on the chosen source and target variables, V2V achieves higher PSNR and SSIM values in the translation task. These results indicate that, unlike V2V, variable pairs selected according to Biswas et al. may not be suitable for variable translation.

To further evaluate the effectiveness of the variable selection process, I use the ionization data set, choose H as the source variable, and translate it to H+, He, He+, and PD. In



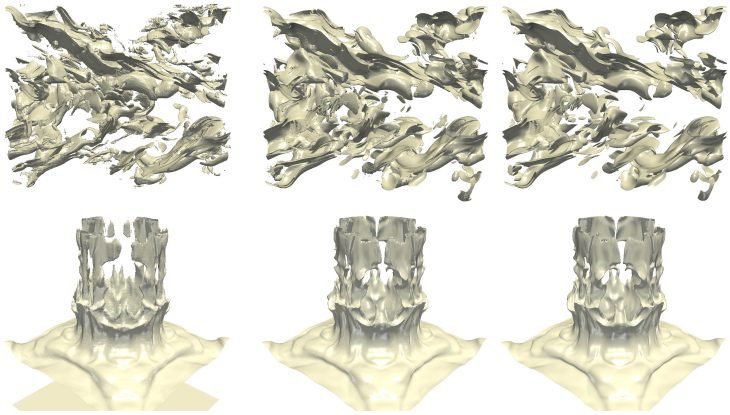
(a) Biswas et al. [13] (b) V2V

Figure 5.9: Comparison of clustered graphs (left column) and translation graphs (right column). Top row: combustion. Bottom row: ionization. For both graphs, the distance between two variables in the 2D graph indicates their similarity.



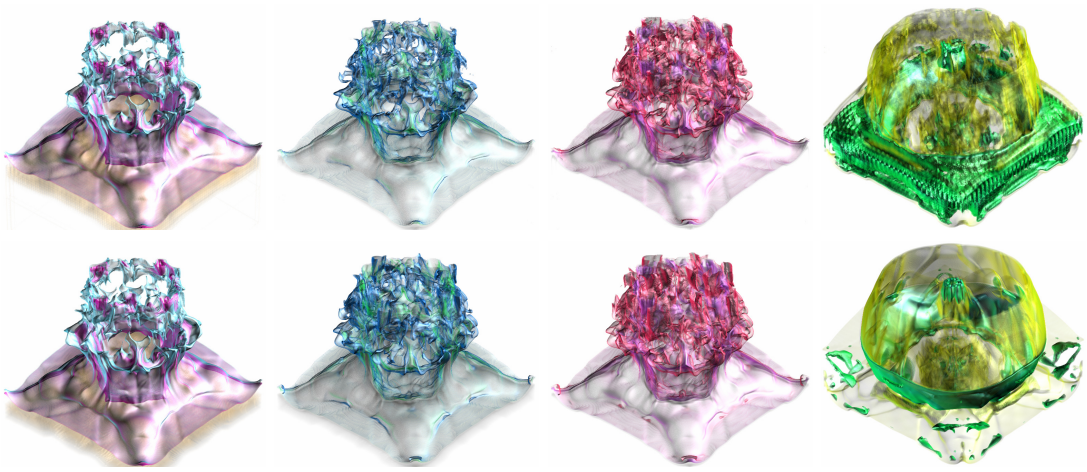
(a) Biswas et al. [13] (b) V2V (c) GT

Figure 5.10: Comparison of variable selection approaches via volume rendering. Variable pairs selected by Biswas et al. are YOH→CHI (top row) and T→H+ (bottom row). Variable pairs selected by V2V are MF→CHI (top row) and H→H+ (bottom row). The displayed time steps are 80 and 50 for CHI and H+, respectively.



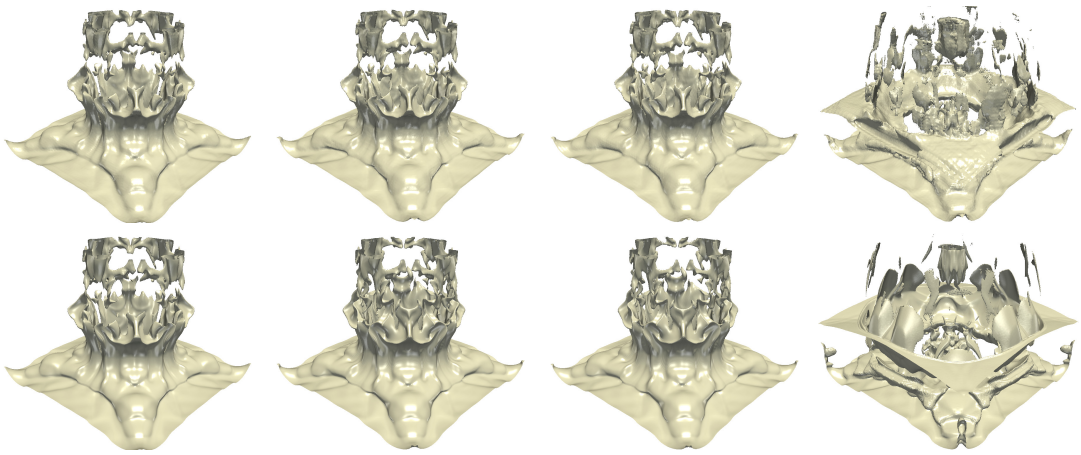
(a) Biswas et al. [13] (b) V2V (c) GT

Figure 5.11: Comparison of variable selection approaches via isosurface rendering. Variable pairs selected by Biswas et al. are YOH→CHI (top row) and T→H+ (bottom row). Variable pairs selected by V2V are MF→CHI (top row) and H→H+ (bottom row). The chosen isovalues are $v = -0.6$ (top row) for CHI and $v = -0.1$ (bottom row) for H+. The displayed time steps are 80 and 50 and for CHI and H+, respectively.



(a) H+ (b) He (c) He+ (d) PD

Figure 5.12: Comparison of volume rendering results of the ionization data set at time step 60. H is chosen as the source variable. Top row: V2V. Bottom row: GT.



(a) H+ (b) He (c) He+ (d) PD

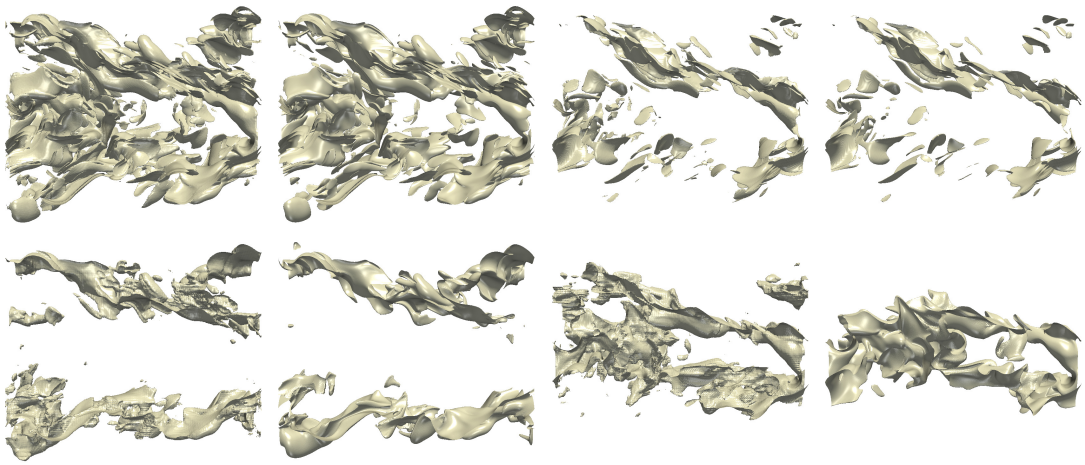
Figure 5.13: Comparison of isosurface rendering results of the ionization data set at time step 60. H is chosen as the source variable. Top row: V2V. Bottom row: GT.

Figure 5.12, I show the volume rendering results. For H→H+, H→He, H→He+, the synthesized results are similar to GT. However, for H→PD, V2V fails to recover the details of PD, particularly, the structure of the top part is not captured. This failure may be explained by a large distance between H and PD shown in the translation graph (Figure 5.9 (b)). The isosurface rendering results are shown in Figure 5.13. For H→H+ and H→He, the isosurfaces generated by V2V are close to GT and almost exhibit the same volumetric features. For H→He+, V2V can still recover the isosurfaces but miss some details. For example, the detailed surface features at the top part are missing in the isosurface synthesized by V2V. For H→PD, V2V fails to generate high-quality isosurface compared with the GT isosurface. For example, the isosurface generated by V2V consists of noises and artifacts, and details are missing at the bottom layer. I also report the average PSNR and SSIM values in Table 5.6. The quantitative results also confirm the difficulty of translating H to PD. Based on the proposed solution, for the combustion set, scientists can save 60 time steps for the variables YOH and CHI if MF is the source variable, and 18.53GB storage is saved in total. As for the ionization, 21.05GB can be saved if H is the source variable since these variables (H+, He, and He+) are only stored 40 time steps.

TABLE 5.6

PERFORMANCE COMPARISON FOR DIFFERENT V2V TRANSLATIONS OF THE IONIZATION DATA SET [2]

target variable	PSNR (dB)	SSIM
H+	45.75	0.951
He	37.44	0.837
He+	39.99	0.874
PD	31.68	0.616

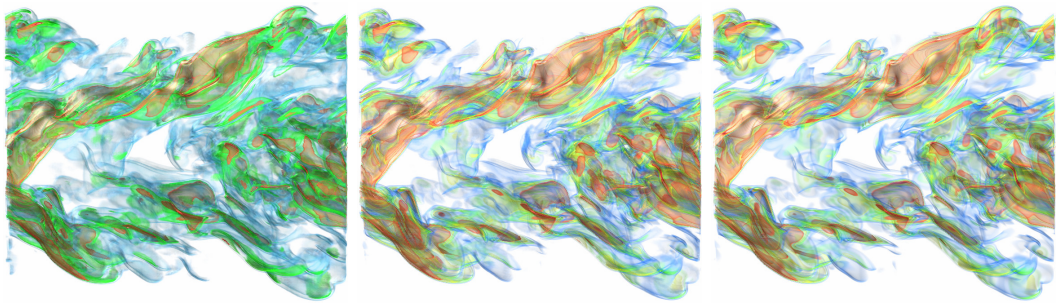


(a) V2V (b) GT (c) V2V (d) GT

Figure 5.14: Evaluation of translation order using the combustion data set via isosurface rendering at time step 72. Top row: MF→CHI (TD = 7.10). Bottom row: CHI→MF (TD = 8.07). The chosen isovalues are $v = -0.6$ (1st and 2nd columns) and $v = 0.5$ (3rd and 4th columns).

Evaluation of variable order. To verify that the translation order does impact the translation performance, I use the combustion data set and choose two translations, MF→CHI and CHI→MF. The results are demonstrated in Figure 5.14. As I can see, CHI→MF is un-

[2] H is the source variable.



(a) V2V w/o FTM (b) V2V (c) GT

Figure 5.15: Comparison of volume rendering results under different architectures using the combustion (MF→CHI) data set at time step 70.

satisfactory since the synthesized isosurfaces fail to capture the interesting features compared with GT. However, MF→CHI is successful since the generated isosurfaces are very close to GT. This asymmetric translation is likely because the essential information in MF is richer than that in CHI, which makes MF→CHI easier than CHI→MF.

Feature translation module. To study what influences the visual quality of the volumes generated by V2V, I conducted such an experiment that trains V2V without using the feature translation module (FTM), i.e., the three purple transformation blocks shown in Figure 5.2 (a). The results are shown in Figure 5.15. I can see that more green content and less yellow content are rendered from the volume generated by V2V without FTM. I speculate that FTM serves the role of refining and filtering the features extracted at different scales during variable translation, improving translation quality.

5.3 Conclusions

I have presented V2V, a new deep learning solution for selecting variables and translating variable sequences for MTVD analysis and visualization. Leveraging GAN, V2V can map one variable sequence to another variable sequence while achieving better visual quality of direct volume rendering and isosurface rendering than HM and two other deep learning solutions (Pix2Pix and CycleGAN). Besides qualitative comparison, quantitative evaluation results using PSNR (data-level), SSIM (image-level), and IS (feature-level) also

confirm the effectiveness of our approach.

V2V can work in the in situ visualization setting. In this scenario, at simulation time, I store the complete sequence for one variable (i.e., all the time steps) while saving the rest of variable sequences sparsely (i.e., only the early time steps) for storage saving. During postprocessing, these reduced variable sequences are synthesized back to their original sequences with high fidelity.

CHAPTER 6

STNET: AN END-TO-END GENERATIVE FRAMEWORK FOR SYNTHESIZING SPATIOTEMPORAL SUPER-RESOLUTION VOLUMES

6.1 STNet

6.1.1 Notation

Let $\mathbf{V}^L = \{\mathbf{V}_1^L, \mathbf{V}_2^L, \mathbf{V}_3^L, \cdots, \mathbf{V}_n^L\}$ and $\mathbf{V}^H = \{\mathbf{V}_1^H, \mathbf{V}_2^H, \mathbf{V}_3^H, \cdots, \mathbf{V}_n^H\}$ be low-resolution and high-resolution time-varying volumetric sequences, respectively, where n denotes the number of time steps. $\mathbf{V}^{L'} = \{\mathbf{V}_1^L, \mathbf{V}_{2+t}^L, \mathbf{V}_{3+2t}^L, \cdots, \mathbf{V}_n^L\}$ is a sparsely sampled low-resolution time-varying volumetric sequence, where t denotes the number of intermediate time steps and $t' = t + 1$ is the *temporal upscaling factor* (i.e., the sequence is upscaled t' times at the temporal dimension). $\mathbf{V}^{L'}$ is also the *pre-training* data in STNet (Figure 6.1 (a)). $\mathbf{V}^T = \{(\mathbf{V}_1^L, \mathbf{V}_1^H), (\mathbf{V}_2^L, \mathbf{V}_2^H), \cdots, (\mathbf{V}_k^L, \mathbf{V}_k^H)\}$ is a sequence used for *fine-tuning* STNet (Figure 6.1 (b)), where k is the total training samples. In this chapter, we set $k = 0.2n$. $\mathbf{V}^S = \{\mathbf{V}_1^S, \mathbf{V}_2^S, \mathbf{V}_3^S, \cdots, \mathbf{V}_n^S\}$ is a super-resolution time-varying volumetric sequence that we aim to generate via STNet. Namely, $\mathbf{V}^H \approx \mathbf{V}^S = \text{STNet}(\mathbf{V}^{L'})$. s is the *spatial upscaling factor* (i.e., each volume is upscaled s times at each spatial dimension). Note that we purposefully refer to the *synthesized* data as *super-resolution* data and the *original* data as *high-resolution* data for differentiation.

6.1.2 Overview

As shown in Figure 6.2, given two-ending low-resolution volumes $\mathbf{V}_{i+t}^L$ and $\mathbf{V}_{i+1+2t}^L$, STNet first leverages $t + 1$ FEI modules to learn features of the intermediate and the two-

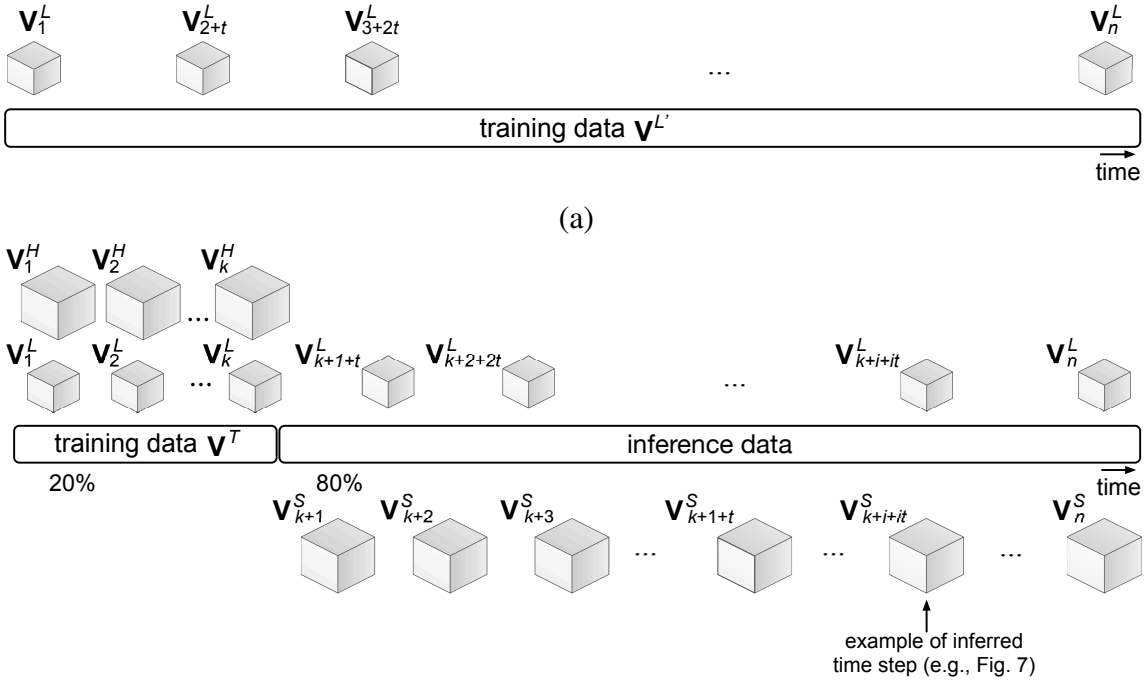


Figure 6.1: Illustration of STNet's training and inference data at (a) pre-training and (b) fine-tuning stages.

ending volumes. One FEI module is tailored for representing the two-ending volumes, and additional t modules are for learning the t intermediate volumes. Once the features are learned, a FU module transforms the spatiotemporal features into a high-dimensional space for generating high-fidelity super-resolution volumes. To discern the spatial and temporal realness of these synthesized volumes, we apply a spatiotemporal discriminator (D) based on *convolutional long short-term memory* (ConvLSTM) [46]. D accepts a volume sequence as input and scores each volume's realness through its spatial (the volume itself) and temporal (its previous time steps) information. To optimize STNet, we propose a two-stage *pre-training* and *fine-tuning* algorithm. During pre-training, we only utilize the low-resolution volumes (i.e., $\mathbf{V}^{L'}$) to optimize STNet using cycle loss. This stage aims to furnish a proper parameter initialization for STNet, which can boost its generalization ability. During fine-tuning, we use $\mathbf{V}^{T}$ as training samples to fine-tune STNet for performance improvement. In the following, we discuss the criteria and rationales for designing

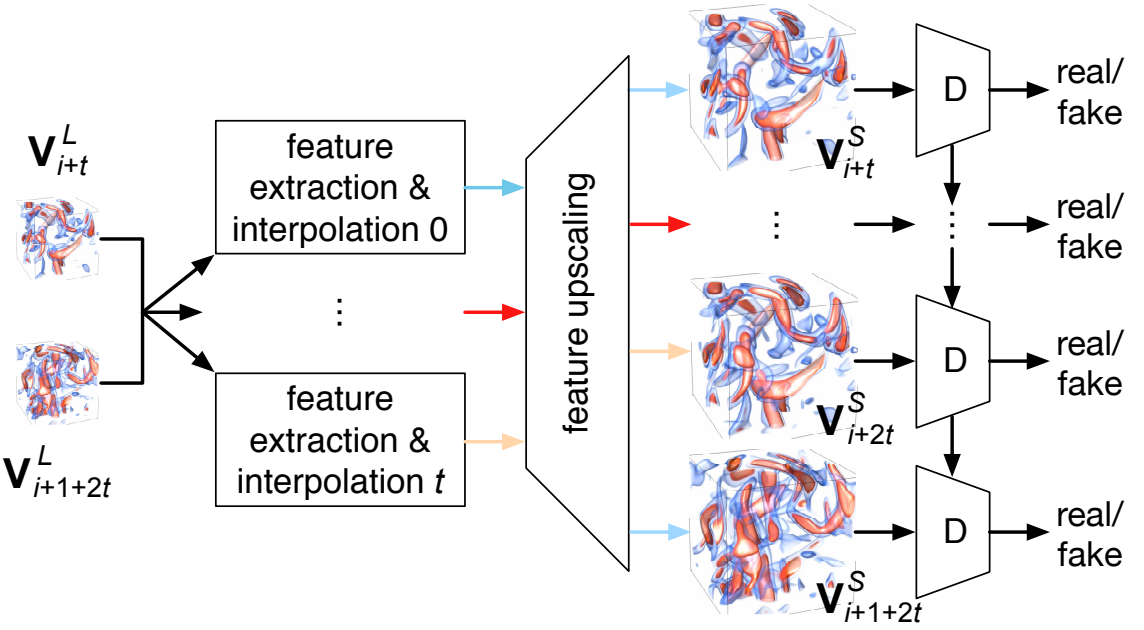


Figure 6.2: Overview of STNet. The network consists of several feature extraction and interpolation (FEI) modules for representing spatiotemporal features and one feature upscaling (FU) module for generating super-resolution volumes. After that, a spatiotemporal discriminator is utilized to discern the spatial and temporal realness.

spatial and temporal modules. Then, we provide optimization details for the pre-training and fine-tuning stages.

6.1.3 Framework

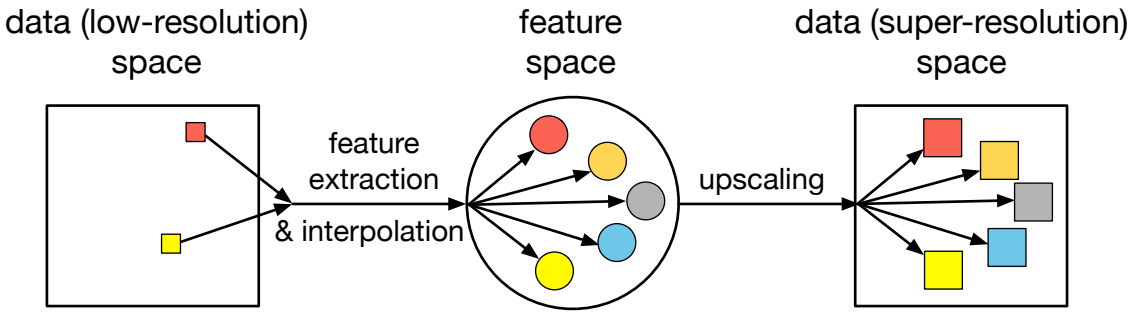
STNet follows a post-upsampling architecture for spatial upscaling and performs interpolation in the feature space for temporal upscaling. The rationales are provided as follows.

Why choose post-upsampling for SSR? Considering SSR architectures, the most widely used ones are *pre-sampling* and *post-upsampling* frameworks [159]. Pre-sampling applies common upscaling approaches (e.g., bicubic and trilinear) for upscaling and follows a series of Conv layers to refine the upscaled data. In contrast, post-sampling leverages Convs to represent low-resolution data and upscales the representations to super-resolution data using deconvolutional or shuffle layers. Compared with pre-sampling, post-

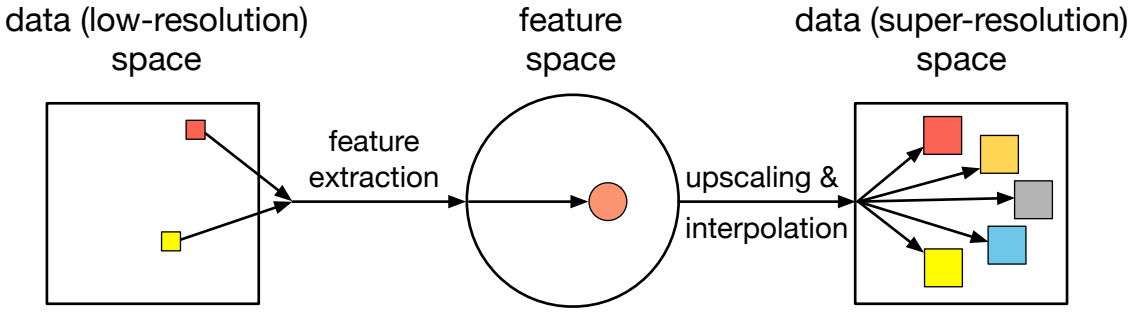
sampling brings two benefits: *speed* and *performance*. First, since most operations perform in the low-dimensional space and only a few operations occur in the high-dimensional space in post-sampling, the computational cost is low. Second, Convs cannot completely eliminate the noises and artifacts introduced by common upscaling approaches in pre-upsampling, while post-upsampling has no such issue in upscaling because Convs already distill data in the low-dimensional space.

Why perform feature-space temporal interpolation? For TSR architectures, two common options are performing interpolation in the *feature* or *data* (super-resolution) space. The feature space refers to the hidden representations of low-resolution volumes, which CNNs usually extract. The data space refers to the space composed of the original volumes. The examples are sketched in Figure 6.3. For feature-space interpolation, give two time steps at both ends, we leverage feature extraction and interpolation to generate the feature of each intermediate time step and the two-ending time steps individually. We then use a FU module to generate the super-resolution volumes from these features. For data-space interpolation, a unified representation is learned from all intermediate and the two-ending time steps. Then the feature is upscaled and interpolated in the data (super-resolution) space. Taking into account the involvement of SSR, feature interpolation is a more suitable solution due to the following reason. Applying data-space interpolation requires a powerful FEI module to learn a representation with rich spatiotemporal information for all intermediate and the two-ending volumes. It also demands a powerful FU module to transform one feature into $t+2$ time steps. This would be difficult, especially in a low-dimensional space, since the information in low-resolution data is limited. For feature-space interpolation, the difficulties of extracting spatiotemporal information and the FU module's demanding capability are mitigated through interpolating multiple features.

Generator. The core of generator lies in the *feature extraction and interpolation* (FEI) and *feature upscaling* (FU) modules. The FEI module comprises four dense blocks



(a) feature-space interpolation



(b) data (super-resolution)-space interpolation

Figure 6.3: Illustration of two different temporal interpolation options.

(DBs) [65]. As shown in Figure 6.4 (a), in each DB, it includes three Conv layers. Each Conv accepts all previous outputs stacked together as input. In particular, we utilize $t+1$ FEI modules to interpolate features of the intermediate and the two-ending volumes. One module accepts the low-resolution volumes as input and produces the corresponding features. The rest of the t modules take the two ending volumes as input and interpolate t features of the intermediate volumes. As sketched in Figure 6.4 (b), in the FU module, we first separate the input into two branches. In each branch, one *voxel shuffle* (VS) layer [46] is used to upscale the input. Then in the second branch, after VS, a Conv and a sigmoid activation function follow. This result is multiplied by the output from the first branch. After merging, two Conv layers and skip connection [65, 126] are utilized to produce the final output [23, 110]. The motivation of using this two-branch-based FU module is that the network can estimate the importance of each neuron in the feature maps, and the more important neurons will offer a larger weight in the following convolution computation. This

design forces the network to pay more attention to interesting volumetric regions instead of treating interesting (e.g., features) and uninteresting (e.g., background) regions equally. We have only a FU module in the generator, which upscales the intermediate features of all time steps. Note that for generating STSR volumes, we need $t+1$ FEI modules: one for representing the two-ending volumes and t for the t intermediate volumes. Rectified linear unit (ReLU) [107] is applied after each Conv layer except the final output layer. No activation function follows after the output layer. Adding tanh or sigmoid will significantly hurt the performance for some specific data sets (e.g., supercurrent) since tanh will saturate at the tails of -1 and 1 and sigmoid will saturate at the tails of 0 and 1, which could kill the gradient and prevent the network from continuous learning.

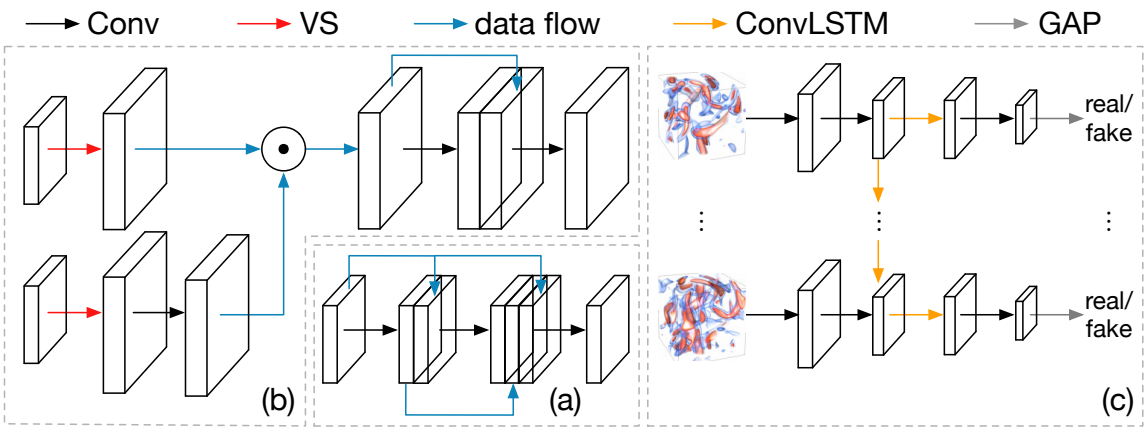


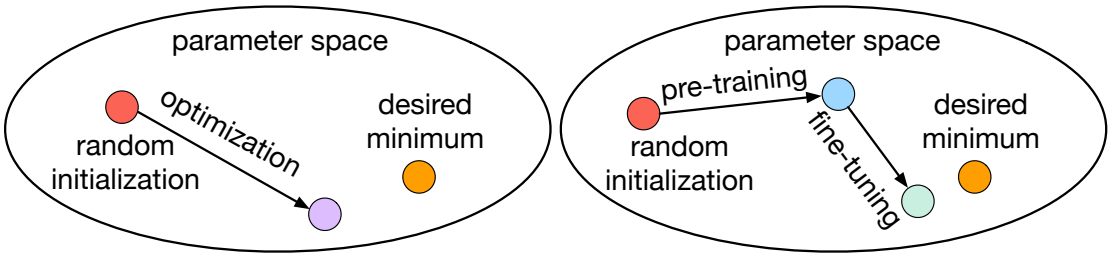
Figure 6.4: Architectures of (a) dense block, (b) FU module, and (c) spatiotemporal discriminator.

Spatiotemporal discriminator. We build a spatiotemporal discriminator to judge the spatial and temporal realness of the volumes generated by STNet. As displayed in Figure 6.4 (c), two Conv layers are utilized to extract spatial information from the input volumes. Each Conv decreases the dimension by half while doubling the channels. Then,

temporal coherence is evaluated by incorporating ConvLSTM that accepts the features of the previous and current time steps as inputs. Finally, a Conv and *global average pooling* (GAP) [90] layer compresses the feature into a single value, which scores the realness of the input volume. ReLU is picked as the activation function, excluding the ConvLSTM and GAP layers.

6.1.4 Optimization

To optimize STNet, we design a two-stage training algorithm: *pre-training* and *fine-tuning*. Pre-training offers an appropriate starting point for training STNet and provides the network with better generalization ability during inference. Fine-tuning fits the network in the downstream STSR task.



(a) w/o pre-training (b) w pre-training

Figure 6.5: Illustration of how learnable parameters change (a) without and (b) with pre-training.

Why pre-training? A straightforward optimization is to randomly initialize the learnable parameters in STNet and directly use low-resolution and high-resolution pairs to train STNet, as sketched in Figure 6.5 (a). Although random initialization can promote the network to a local minimum, it could not improve further to the desired one since it would overfit the training data. However, adding pre-training can promote the randomly initial-

ized parameters to a local minimum based on the pre-training task, which can be utilized in the downstream task. Then, based on this new starting point, the network can better fit the data, as shown in Figure 6.5 (b). The pre-training stage can guide the learning process towards the minima to support better generalization from the training data [30]. In the context of STSR, pre-training can help STNet see the inference data in the low-dimensional space, preventing the network from overfitting in the training data and enhancing the network's generalization ability in the inference data.

Pre-training. To pre-train in an unsupervised fashion, we leverage *cycle* loss to optimize STNet. The cycle loss for $\mathbf{V}_i^S$ is defined as

$$\mathscr{L}_{\text{cyc}} = ||\mathscr{D}(\mathbf{V}_i^S) - \mathbf{V}_i^L||_2, \tag{6.1}$$

where $\mathscr{D}$ denotes a downsizing operation (e.g., trilinear) and $||\cdot||_2$ is L^2 norm. The rationale for designing this loss is that once the super-resolution volumes are generated and if we downsize them again to the low-dimensional space, the downsized version (i.e., $\mathscr{D}(\mathbf{V}_i^S)$) should be consistent with the original low-resolution volumes (i.e., $\mathbf{V}_i^L$).

Fine-tuning. To fine-tune STNet in the STSR task, we leverage *volumetric* loss for the closeness to high-resolution volumes, and *adversarial* loss for the realness, to train STNet. The volumetric loss for $\mathbf{V}_i^S$ is defined as

$$\mathscr{L}_{\text{vol}}^{\text{G}} = ||\mathbf{V}_i^S - \mathbf{V}_i^H||_2. \tag{6.2}$$

The adversarial losses of generator G and discriminator D for $\{\mathbf{V}_i^S, \cdots, \mathbf{V}_{i+t+1}^S\}$ are defined as

$$\mathscr{L}_{\text{adv}}^{\text{G}} = \sum_{j=0}^{t+1} 1 - \text{D}\left(\mathbf{V}_{i+j}^S | \mathbf{V}_{i+j-1}^S, \cdots, \mathbf{V}_i^S\right), \tag{6.3}$$

$$\begin{aligned}\mathscr{L}_{\text{adv}}^{\text{D}} = &\sum_{j=0}^{t+1} \text{D}\left(\mathbf{V}_{i+j}^{S} | \mathbf{V}_{i+j-1}^{S}, \cdots, \mathbf{V}_{i}^{S}\right) \\ &+ \sum_{j=0}^{t+1} 1 - \text{D}\left(\mathbf{V}_{i+j}^{H} | \mathbf{V}_{i+j-1}^{H}, \cdots, \mathbf{V}_{i}^{H}\right).\end{aligned} \tag{6.4}$$

Considering both volumetric and adversarial losses, we define the total loss of G as

$$\mathscr{L}^{\text{G}} = \lambda_{\text{vol}} \mathscr{L}_{\text{vol}}^{\text{G}} + \lambda_{\text{adv}} \mathscr{L}_{\text{adv}}^{\text{G}}, \tag{6.5}$$

where λ_{vol} and λ_{adv} control the weights of these two losses.

The training algorithm of STNet is listed in Algorithm 1. In the pre-training stage, we first use the sparsely sampled low-resolution sequence $\mathbf{V}^{L'}$, as sketched in Figure 6.1 (a), to train STNet. After training T_P epochs, we begin to fine-tune STNet using the low-resolution and high-resolution pairs at the early time steps $\mathbf{V}^{T}$, as shown in Figure 6.1 (b). Following Wang et al. [157], the fine-tuning stage contains two steps. First, only volumetric loss is applied to optimize STNet for training stabilization and computational cost reduction. Second, the spatiotemporal discriminator D is involved in the training procedure for enhancing spatial and temporal coherence.

6.2 Results and Discussion

6.2.1 Data Sets and Network Training

We tested STNet using the data sets reported in Table 6.1. Note that half-cylinder is an ensemble data set with different Reynolds numbers (i.e., 320, 640, and $6,400$). PyTorch [113] was used for implementation. Training and inference were performed on an NVIDIA TESLA V100 GPU. The low-resolution data were obtained by applying bicubic kernel with reflection padding. We scaled the range of $\mathbf{V}^{L}$ and $\mathbf{V}^{H}$ to $[-1,1]$. We initialized parameters following He et al. [57] for optimization and utilized the Adam optimizer [78]

Algorithm 2: STNet training algorithm.

Require: initial parameters θ_{G} and θ_{D} for G and D, number of training epochs in pre-training and fine-tuning stages: T_P, T_{F_1}, and T_{F_2}, learning rates α_{G} and α_{D} for G and D, respectively.

```
/* pre-training stage */
for j = 1 ··· T_P do
    sample low-resolution data from V^{L'};
    compute L_cyc according to Equation 6.1;
    update θ_G;
end for
/* fine-tuning stage 1: only using volumetric loss to optimize STNet */
for j = 1 ··· T_{F_1} do
    sample low-resolution and high-resolution data pairs from V^T;
    compute L_vol^G according to Equation 6.2;
    update θ_G;
end for
/* fine-tuning stage 2: taking temporal coherence into consideration */
for j = 1 ··· T_{F_2} do
    sample low-resolution and high-resolution data pairs from V^T;
    compute L_adv^D according to Equation 6.4;
    update θ_D;
    compute L^G according to Equation 6.5;
    update θ_G;
end for
```

for parameter update. In each mini-batch, one training sample is used. The learning rates for G and D are 10^{-4} with $\beta_1 = 0.9$, $\beta_2 = 0.999$. $\lambda_{\mathrm{vol}} = 1$ and $\lambda_{\mathrm{adv}} = 10^{-3}$. T_P, T_{F_1}, and T_{F_2} are set to 200, 400, and 50 epochs, respectively, for all data sets. All these hyperparameter settings are determined based on experiments.

6.2.2 Results

Baselines. We compare STNet with three baseline solutions:

- BL: Bicubic interpolation is used for SSR and linear interpolation for TSR. BL stands for bicubic+linear interpolation.
- SSR+TSR: SSR [45] is a GAN solution for time-varying data SSR, and TSR [46] is a recurrent generative solution for TSR. We train SSR for 400 epochs and TSR for 400 epochs.
- STD: STD is a variant of STNet. Instead of performing temporal interpolation in the feature space, STD directly interpolates the volumes in the data space. Namely, given two volumes at both ends, STD leverages a FEI module to simultaneously learn spatiotemporal features of all intermediate and the two-ending time steps and applies a FU module to generate super-resolution volumes.

TABLE 6.1

VARIABLES AND DIMENSION OF EACH DATA SET

data set	variables	dimension ($x \times y \times z \times t$)
five jets	intensity	$128 \times 128 \times 128 \times 100$
ionization	H, H+, He, He+	$600 \times 248 \times 248 \times 100$
half-cylinder [125]	velocity magnitude	$640 \times 240 \times 80 \times 100$
supercurrent	rho	$256 \times 128 \times 32 \times 200$
Tangaroa [116]	velocity magnitude	$300 \times 180 \times 120 \times 150$
vortex	vorticity magnitude	$128 \times 128 \times 128 \times 90$

Note that existing STSR works [106, 133, 135, 141] are not suitable for 3D data since they could not capture complicated spatiotemporal patterns. As for Xiang et al. [166], it leverages deformable Conv for spatiotemporal super-resolution. However, it is difficult to extend this architecture to handle 3D data sets for two reasons. First, deformable Conv needs to learn offsets to perform Convs, which requires additional parameters and memories. Second, the offsets are learned from the whole data, not a subregion. Thus, the computational cost is extremely high when the volume is large.

For the same data set, all visualization results follow the same rendering parameters for lighting, viewpoint, transfer function (used in volume rendering), and isovalue (used in isosurface rendering). Except for the GT results, all results from STNet and baseline solutions are rendered using inferred data from a later time step (refer to Figure 6.1 (b)).

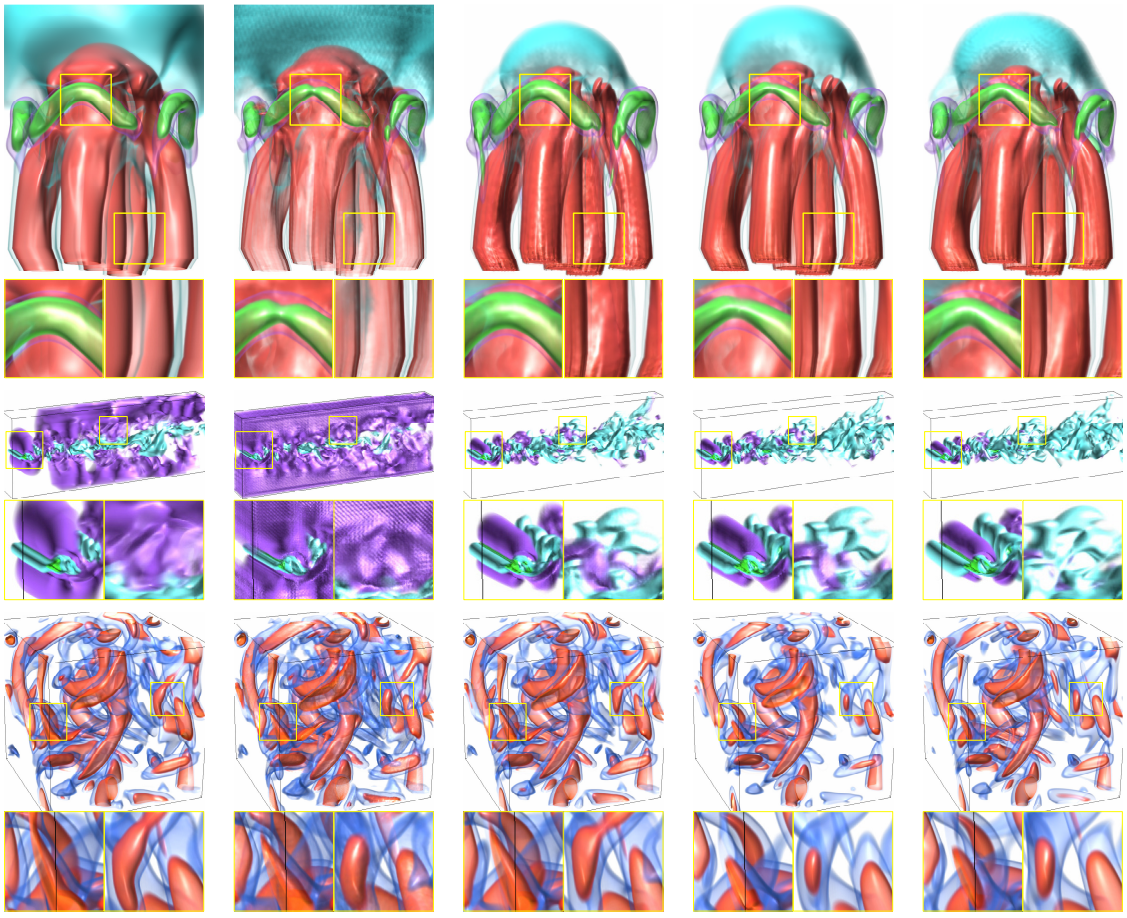
Evaluation metrics. We utilize three metrics, including data-level *peak signal-to-noise* (PSNR), image-level *structural similarity index* (SSIM), and feature-level *isosurface similarity* (IS) [16], for quantitative evaluation.

Quantitative and qualitative analysis. Figure 6.6 shows volume rendering results produced from BL, SSR+TSR, STD, STNet, and GT using the five jets, half-cylinder (640),

TABLE 6.2

PERFORMANCE AND TIME COMPARISON

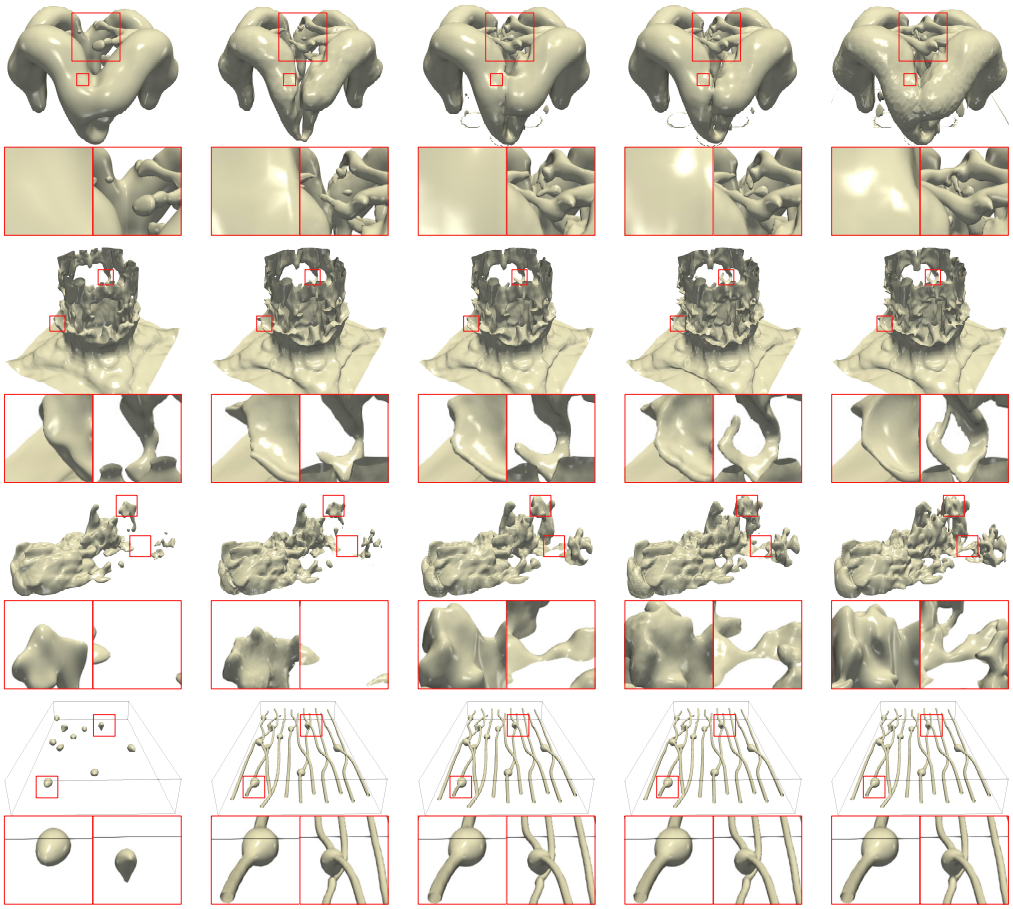
data set	method	PSNR	SSIM	training time
five jets	BL	27.96	0.751	—
	SSR+TSR	27.30	0.685	64.024
	STD	**40.00**	0.892	24.662
	STNet	39.63	**0.901**	43.702
supercurrent	BL	26.92	0.812	—
	SSR+TSR	44.54	0.995	87.258
	STD	**44.74**	0.995	31.443
	STNet	44.71	**0.995**	66.285
half-cylinder (640)	BL	28.12	0.864	—
	SSR+TSR	24.15	0.792	66.187
	STD	35.60	0.907	27.884
	STNet	**36.84**	**0.944**	45.580
Tangaroa	BL	21.85	0.853	—
	SSR+TSR	26.96	0.858	96.037
	STD	30.07	0.879	37.964
	STNet	**33.26**	**0.892**	65.568
ionization (H)	BL	33.52	0.862	—
	SSR+TSR	43.05	0.908	68.123
	STD	40.67	0.892	28.556
	STNet	**43.19**	**0.913**	44.781
vortex	BL	29.78	**0.749**	—
	SSR+TSR	23.89	0.576	57.622
	STD	31.12	0.693	24.573
	STNet	101**32.73**	0.720	39.329



(a) BL (b) SSR+TSR (c) STD (d) STNet (e) GT

Figure 6.6: Comparison of volume rendering results. Top to bottom: five jets, half-cylinder (640), and vortex.

and vortex data sets. For the five jets data set, both BL and SSR+TSR produce more cyan parts at the cap, and the rendering results are overly smooth at the legs. STD and STNet generate similar results, but taking a close comparison, STNet synthesizes finer details at the green part (refer to the zoom-ins on the left) and the legs (refer to the zoom-ins on the right). For the half-cylinder (640) data set, both BL and SSR+TSR do not produce high-quality rendering results. STD generates the result with tiny noises and artifacts at the front (refer to the zoom-ins on the left) and more purple parts (refer to the zoom-ins on the right). As for the vortex data set, BL, SSR+TSR, and STD produce more blue and red parts over the whole volume. STNet generates closer results compared with GT. For



(a) BL (b) SSR+TSR (c) STD (d) STNet (e) GT

Figure 6.7: Comparison of isosurface rendering results. Top to bottom: five jets, ionization (H), Tangaroa, and supercurrent. The chosen isovalues are 0, 0.5, 0, and -0.2 , respectively.

the quantitative results, we report average PSNR and SSIM values in Table 6.2. In general, STNet achieves the best performance among these four solutions, except for the average SSIM for the vortex data set and the average PSNR for the five jets and supercurrent data sets. The model sizes of SSR+TSR, STD, and STNet are 74.0MB, 138MB, and 62.5MB, respectively. As for the training time, STD requires the shortest time for optimization, and SSR+TSR needs the longest time. This is because SSR has one generator and two discriminators, and TSR has a recurrent generator and one discriminator. SSR+TSR needs to optimize two generators and three discriminators to go through one training data sample, incurring an expensive computational cost. Since STNet has more FU modules than STD

($t+1$ vs. 1), it demands more time to compute gradients and optimize. However, there is no significant difference for the inference time.

Figure 6.7 shows isosurface rendering results among BL, SSR+TSR, STD, STNet, and GT using the five jets, ionization (H), Tangaroa, and supercurrent data sets. For each data set, we pick one isovalue for comparison. For the five jets data set, both BL and SSR+TSR do not capture the isosurface details (refer to the zoom-ins on the right), and the lighting on the isosurface generated by STD shifts too much compared with GT (refer to the zoom-ins on the left). For the ionization (H) data set, STNet can capture finer structures (refer to the zooms-in on the right) compared with other solutions. For the Tangaroa data set, both BL and SSR+TSR do not produce isosurfaces with fine details. In addition, the isosurfaces generated by SSR+TSR contain noticeable noises and artifacts. For STD and STNet, both can synthesize similar isosurfaces compared with GT. However, STNet can extract more details. For example, it produces close isosurfaces at two corners (refer to the zoom-ins). For the supercurrent data set, BL does not extract close isosurfaces compared with GT, while SST+TSR, STD, and STNet produce similar results, and all of them are comparable to GT. In terms of quantitative comparison, Table 6.3 reports the average IS score for BL, SSR+TSR, STD, and STNet. STNet achieves the best performance for all data sets. Note that for the five jets and supercurrent data sets, STD and STNet achieve similar performance. This is because the overall content does not change too much over different time steps for these two data sets (i.e., the training and inference data are similar), which means data-space interpolation could lead to satisfactory results. But to achieve similar performance, STD needs around 36 million parameters while STNet requires about 16 million.

Comparison with baselines. As shown in Figures 6.6 and 6.7, STNet outperforms SSR+TSR and STD in terms of visual quality and achieves better quantitative scores for most data sets compared with STD. The potential reasons are as follows. SSR+TSR directly uses two networks to perform SSR and TSR, respectively. It forces the latter network

TABLE 6.3

AVERAGE IS VALUES AT CHOSEN ISOVALUES

data set (isovalue)	BL	SSR+TSR	STD	STNet
five jets ($v = 0$)	0.78	0.80	**0.88**	**0.88**
half-cylinder (640) ($v = 0.2$)	0.62	0.60	0.74	**0.79**
ionization (H) (v =0.5)	0.71	0.78	0.79	**0.81**
supercurrent ($v = -0.2$)	0.23	**0.96**	**0.96**	**0.96**
Tangaroa ($v = 0$)	0.57	0.59	0.73	**0.75**
vortex ($v = -0.2$)	0.82	0.79	0.84	**0.86**

(i.e., TSR) to complete two tasks, i.e., TSR and denoising, since the results generated from the former network (i.e., SSR) are not GT, and they contain unobservable noises. These noises could be sensitive [61, 179] when synthesizing high-quality volumes. This explains why the rendering results produced by SSR+TSR contain noises and artifacts. We try to add a denoising module into the TSR framework to clean up these noises, but the results are not satisfactory. For STD, it achieves comparable PSNR values for simple data sets (e.g., supercurrent) but cannot generate high fidelity results for complex data sets (e.g., half-cylinder). This is because extracting a global spatiotemporal representation for all intermediate and the two-ending time steps is extremely difficult for these volumes whose patterns change dynamically. We point out that the number of parameters in STD is twice of those in STNet. The reason is as follows. Ideally, to achieve a fair comparison between STD and STNet, we need to set the same number of parameters in both STD and STNet. However, under the model size of 62.5MB, STD cannot generate satisfactory STSR volumes. Therefore, we expand the width (i.e., the number of channels) of STD.

Comparison with state-of-the-art compression. Figure 6.8 shows volume rendering results obtained from the upscaled volumes generated by STNet and the volumes com-

TABLE 6.4

COMPARISON OF TTHRESH AND STNET [1]

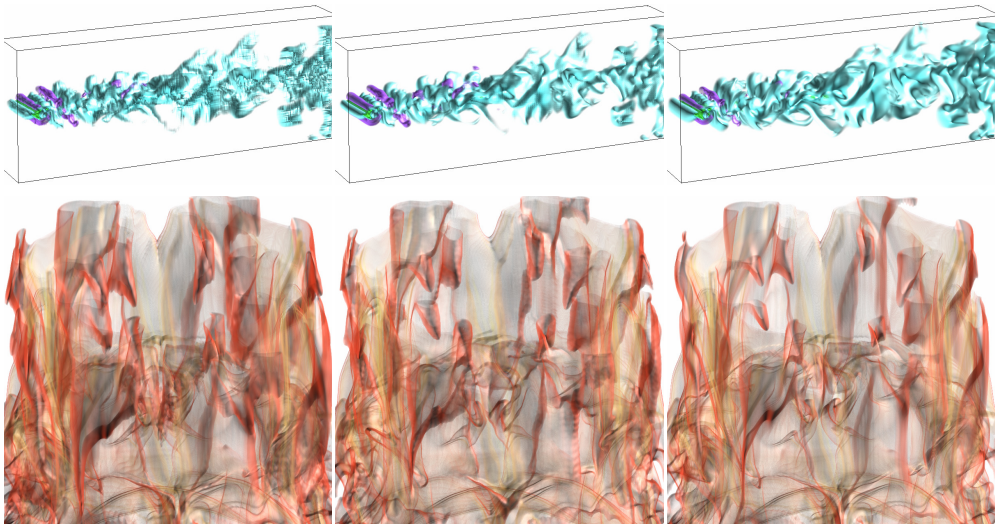
data set	method	compression ratio	SSIM
half-cylinder (640)	TTHRESH	**1745.33** ×	0.902
	STNet	162.62 ×	**0.944**

data set	method	PSNR	SSIM
ionization (H)	TTHRESH	**49.37**	0.906
	STNet	43.19	**0.913**

pressed then decompressed using TTHRESH [5]. We choose TTHRESH, a tensor compression solution, because it smoothly degrades the data, leads to errors smaller than other state-of-the-art algorithms, and requires a low cost for compression and decompression. We consider two scenarios: (1) keeping the same PSNR (i.e., 36.84 dB) for the half-cylinder (640) data set and (2) controlling the same compression ratio (i.e., 206.74×) for the ionization (H) data set. To achieve a fair comparison against TTHRESH, we include the model size in the computation of compression ratio. We utilize a lossless compression algorithm [91] to further reduce the storage of the saved model and data. For the first scenario, as shown in Figure 6.8 (a), the image rendered by TTHRESH contains fewer cyan parts. It also produces noticeable noises in the rendering image. The top part of Table 6.4 reports the compression ratios and average SSIM values for both methods. Under the same PSNR, although TTHRESH achieves a higher compression ratio, which is about 10 times compared with that of STNet, STNet achieves a higher SSIM value for the syn-

[1] Top: average SSIM under the same PSNR of 36.84 dB. Bottom: average PSNR (dB) and SSIM under the same compression ratio of 206.74×.

thesized volumes than those recovered from TTHRESH. For the second scenario, as shown in Figure 6.8 (b), TTHRESH generates more red parts. The bottom part of Table 6.4 gives average PSNR and SSIM values for both methods. Under the same compression ratio, although TTHRESH produces a higher PSNR value, STNet achieves a higher SSIM value and better visual quality.

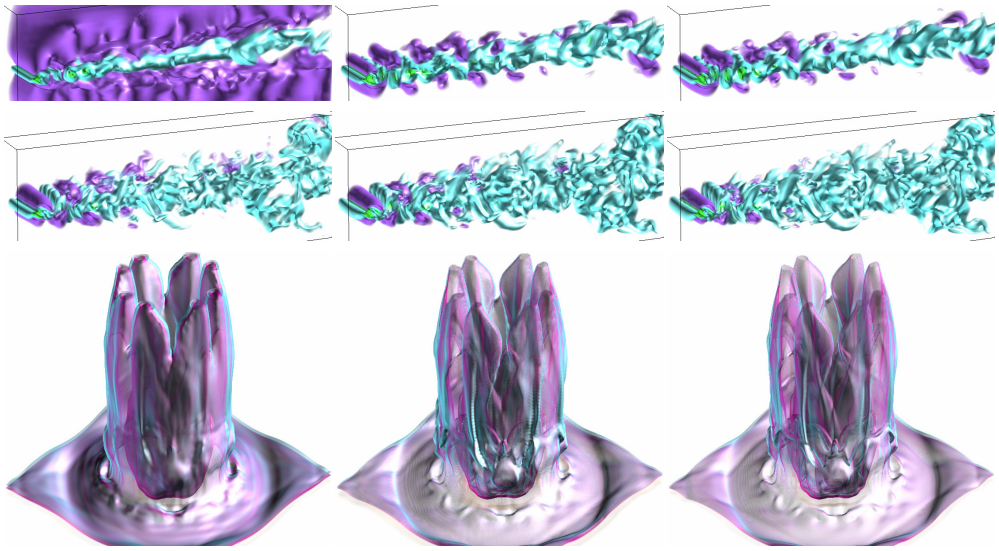


(a) TTHRESH (b) STNet (c) GT

Figure 6.8: Volume rendering results. Top and bottom: half-cylinder (640) and ionization (H).

Evaluation of ensemble and multivariate data sets. In Figures 6.9 and 6.10, we compare volume and isosurface rendering results from the synthesized volumes given by BL and STNet on ensemble and multivariate data sets to evaluate the generalization ability. We use an ensemble parameter (variable) X_S of a data set for training, while another ensemble parameter (variable) X_T of the same data set is used for inference (i.e., $X_S \rightarrow X_T$). For the half-cylinder data set, we test two cases: $640 \rightarrow 320$ and $640 \rightarrow 6,400$. The complexity increases as the Reynolds number gets large. For $640 \rightarrow 320$, compared with the result gen-

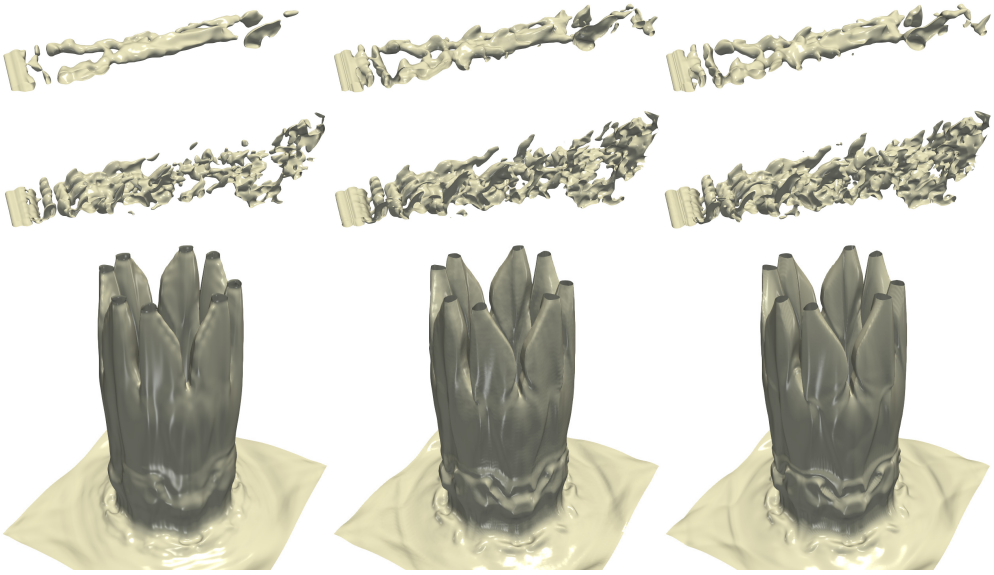
erated by BL, STNet produces better visual results in the purple and cyan parts of volume rendering results and extracts finer details of isosurfaces as shown in isosurface rendering results. For $640 \rightarrow 6,400$, STNet synthesizes more detailed rendering results. For example, the cyan part's lighting and the isosurfaces at the middle and right corners are closer to GT results. As for H $\rightarrow$ H+ of the ionization data set, BL produces fewer details at the bottom of the ionization for both rendering results. For instance, for volume rendering, the image generated by BL shows more purple color. For isosurface rendering, the lighting at the bottom generated by BL is inconsistent with that of GT. As for quantitative results, STNet also outperforms BL in terms of PSNR and SSIM, as shown in Table 6.5.



(a) BL (b) STNet (c) GT

Figure 6.9: Variable and ensemble volume rendering results. Top to bottom: half-cylinder (320), half-cylinder (6,400), and ionization (H+).

Evaluation of s and t. To analyze the performance of STNet with different s and t, we set $s = 4$ and $s = 8$ with different t using the five jets and supercurrent data sets, respectively. As shown in the top rows of Figures 6.11 and 6.12, $t = 3$ achieves the best



(a) BL (b) STNet (c) GT

Figure 6.10: Variable and ensemble isosurface rendering results. Top to bottom: half-cylinder (320), half-cylinder (6,400), and ionization (H+). The chosen isovalues are -0.2, 0, and -0.2, respectively.

quality. However, all of them can capture the overall shape and details of the five jets, and the main difference is the size of the cyan cap. For the supercurrent data set, the rendering results are shown in the bottom rows of Figures 6.11 and 6.12. All produce similar results compared to GT for volume and isosurface rendering. But taking a close comparison, under $t = 10$, the isosurface is broken into two parts at the top-left corner (refer to the red arrow). Besides, average PSNR and SSIM values are shown in Figure 6.13. STNet significantly outperforms BL for the five jets and supercurrent data sets under different settings of s and t.

Based on the above results, our suggestions for choosing s and t for different data sets are as follows.

- With $s = 4$, the appropriate value for t could be large for simple data sets (e.g., five jets and supercurrent), where the patterns change slowly over time. The suitable value is determined by the total sample time steps. With sufficient samples, t could be 9 for the supercurrent data set (200 time steps), while with limited samples, t could be 5 for the five jets data set (100 time steps).
- With $s = 4$, for complex data sets (e.g., half-cylinder and vortex) where the patterns

TABLE 6.5

PERFORMANCE COMPARISON FOR ENSEMBLE AND MULTIVARIATE DATA SETS

data set ($X_S \rightarrow X_T$)	method	PSNR	SSIM
half-cylinder ($640 \rightarrow 320$)	BL	30.01	0.886
	STNet	**35.26**	**0.951**
half-cylinder ($640 \rightarrow 6,400$)	BL	26.47	0.857
	STNet	**33.86**	**0.926**
ionization (H→H+)	BL	33.53	0.867
	STNet	**42.99**	**0.910**

could evolve rapidly in neighborhood time steps, 3 is a proper value for t.

- With $s = 8$, it is almost infeasible with the current architecture for upscaling these data sets (e.g., half-cylinder and vortex), where the spatial structures are complex.
- For data sets (e.g., five jets and supercurrent) where the shapes are simple and less complicated, $s = 8$ could still work.

Temporal coherence. To compare how well temporal coherence is preserved using BL and STNet, we show five consecutive time steps using the half-cylinder (320) data set. As shown in Figure 6.14, STNet can better capture temporal coherence compared with BL as BL does not produce meaningful intermediate time steps. This is because BL only assumes that features evolve linearly, which is not the case for most data sets.

6.2.3 Network Analysis

To analyze STNet, we study the impact of pre-training and loss function. A detailed discussion is as follows.

Evaluation of pre-training. To investigate the effectiveness of adding pre-training, we train STNet with and without pre-training. Table 6.6 gives the average PSNR and SSIM

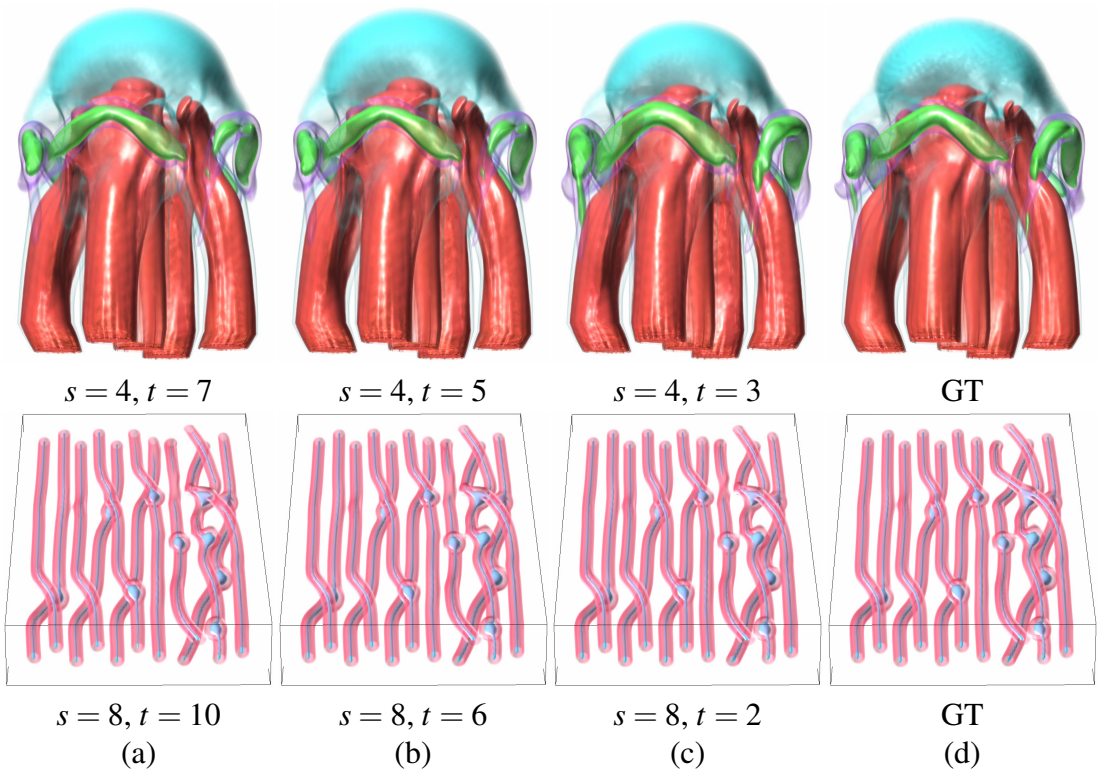


Figure 6.11: Volume rendering results under different s and t. Top and bottom: five jets and supercurrent.

under these two training schemes. As we can see, the pre-training can improve about 1dB and 0.01 for the average PSNR and SSIM values, respectively. Moreover, we plot the PSNR curves over the whole sequence, as shown in Figure 6.15. The curves indicate that pre-training can boost the PSNR value at almost every time step. As for visual quality, volume rendering results are shown in the top row of Figure 6.16. Clearly, using pre-training can generate results closer to GT (refer to the yellow ellipse). These quantitative and qualitative analysis results confirm the usefulness of the pre-training algorithm.

Evaluation of volumetric loss. Cycle and volumetric losses serve a similar role in optimization while constraining the volumes in the low-dimensional and high-dimensional spaces, respectively. So, would it still work if we only leverage one loss to train the network? To answer this question, we optimize the network with and without considering volumetric loss. Note that training without cycle loss means removing pre-training, which has

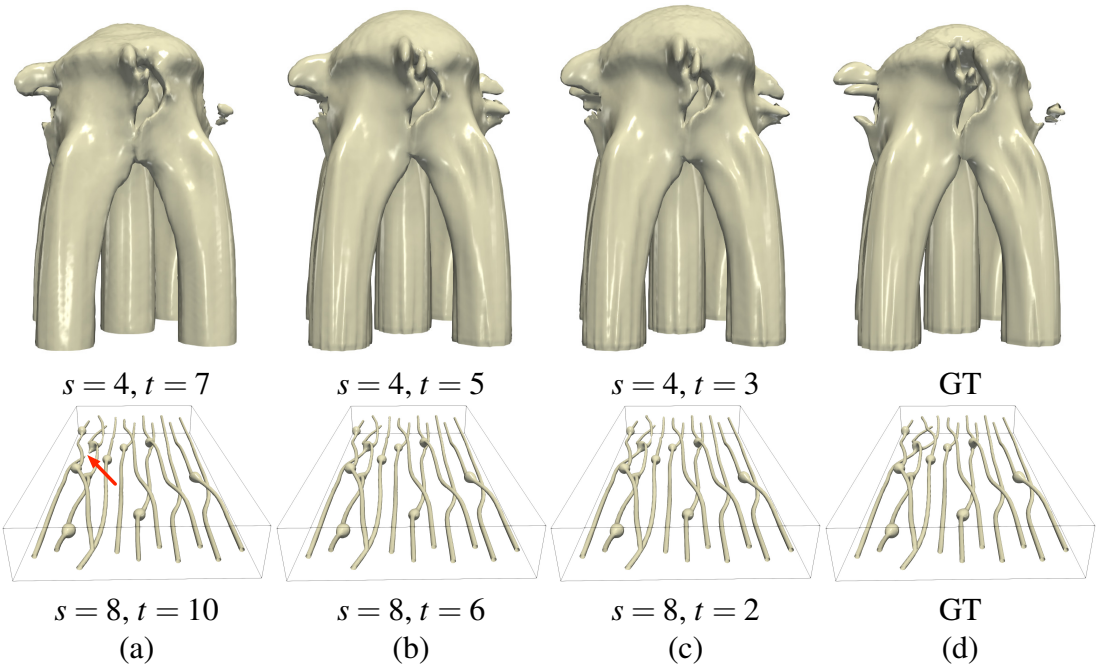
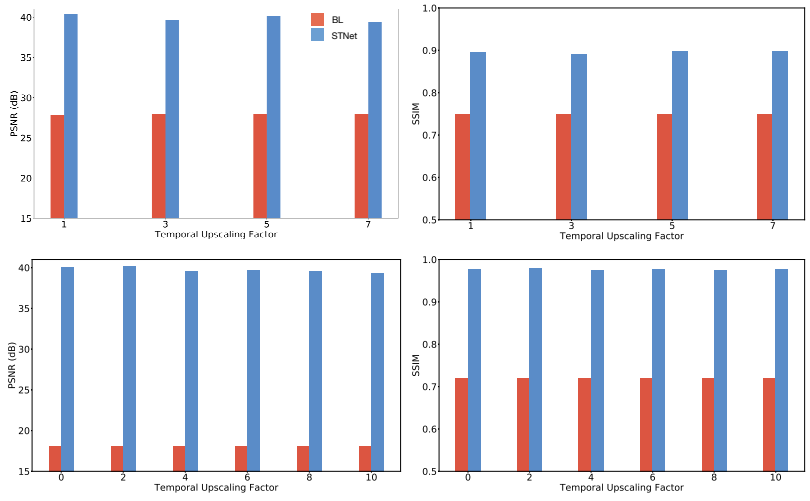
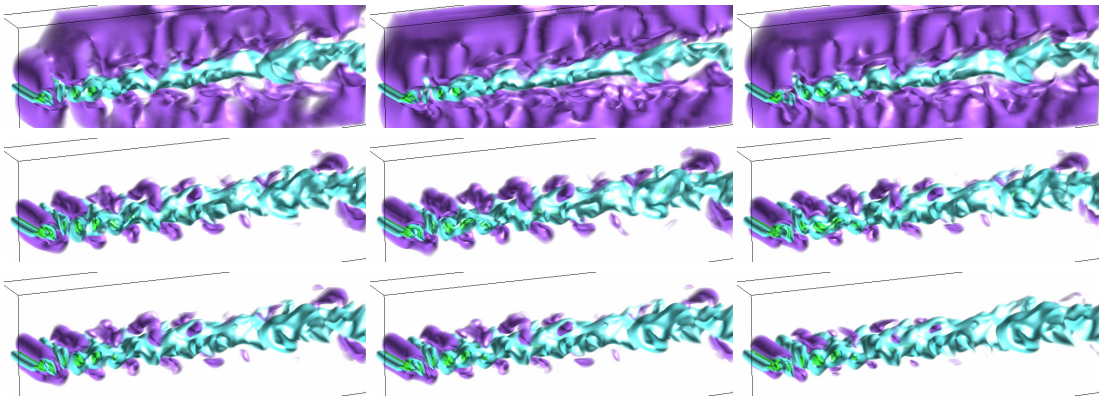


Figure 6.12: Isosurface rendering results under different s and t. Top and bottom: five jets and supercurrent. The chosen isovalues are 0.4 and -0.2, respectively.



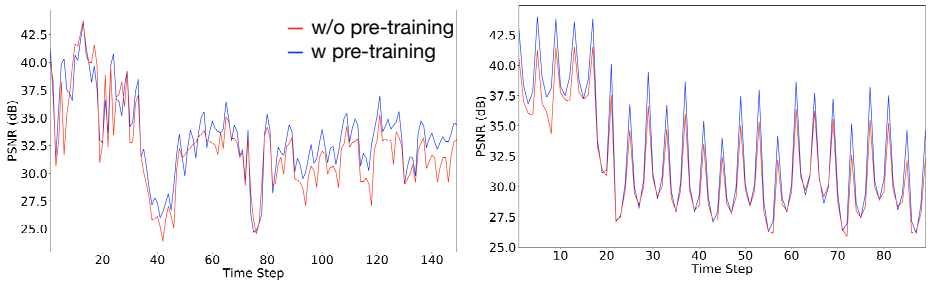
(a) PSNR (b) SSIM

Figure 6.13: Average PSNR and SSIM under different s and t. Top and bottom: five jets with $s = 4$ and supercurrent with $s = 8$.



(a) 94 (b) 95 (c) 96

Figure 6.14: Volume rendering results of the half-cylinder (320) data set with five time steps (94 to 96). Top to bottom: BL, STNet, and GT.



(a) Tangaroa (b) vortex

Figure 6.15: PSNR curves with and without pre-training.

been discussed. As shown in Table 6.6, without volumetric loss, average PSNR and SSIM values drop significantly. As for rendering quality, we display volume rendering results in Figure 6.16. Using cycle loss only captures the overall shape of the five jets but could not preserve fine details. This is because different data samples in the high-dimensional space can be downsized to the same data in the low-dimensional space with the same downsizing function (e.g., bicubic) [101]. Constrained only in the low-dimensional space, the network could jump into an undesired local minimum in the high-dimensional space.

Evaluation of adversarial loss. To study the impact of adversarial loss, we optimize STNet with and without adversarial loss. Table 6.6 reports the average PSNR and SSIM under these two optimizations. Although we can achieve a higher PSNR value without ad-

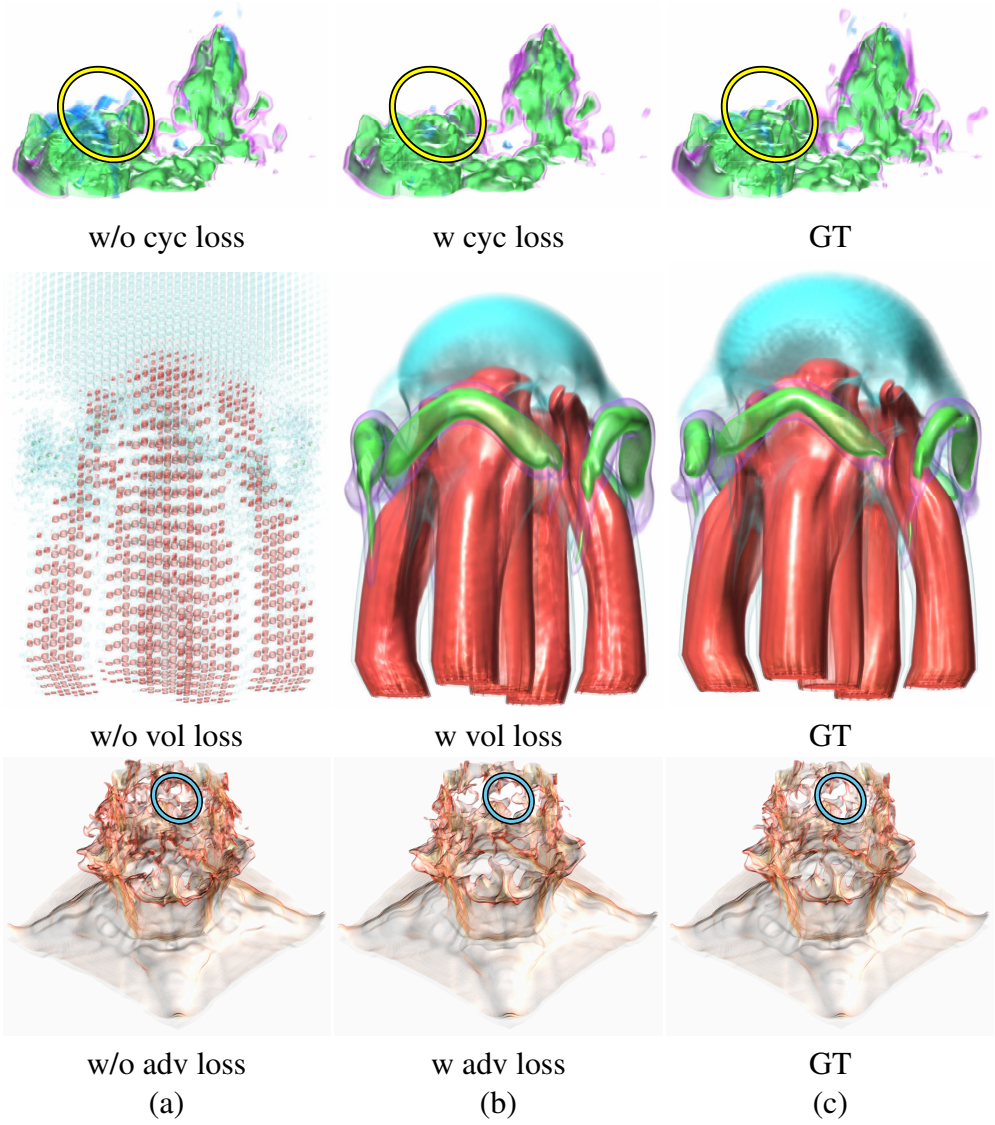


Figure 6.16: Volume rendering results under different loss settings. From top to bottom: Tangaroa, five jets, and ionization (H).

TABLE 6.6

PERFORMANCE COMPARISON UNDER DIFFERENT SETTINGS

data set	method	PSNR	SSIM
Tangaroa	w/o pre-training	32.26	0.883
	w pre-training	**33.26**	**0.892**
vortex	w/o pre-training	31.75	0.709
	w pre-training	**32.73**	**0.720**
five jets	w/o volumetric loss	22.11	0.621
	w volumetric loss	**39.63**	**0.892**
half-cylinder (640)	w/o volumetric loss	20.62	0.888
	w volumetric loss	**36.84**	**0.944**
ionization (H)	w/o adversarial loss	**43.80**	0.904
	w adversarial loss	43.19	**0.913**

versarial loss, adding adversarial loss improves SSIM values (i.e., the image-level metric). Moreover, the rendering images also confirm that adversarial loss can boost perceptual quality, as shown in Figure 6.16. For example, without adversarial loss, the rendering image produces more red parts at the ionization's head. Therefore, these results demonstrate the usefulness of adversarial loss in improving visual quality.

6.3 Conclusions

I have presented STNet, a novel generative solution for producing STSR volumes for time-varying data analysis and visualization. Leveraging post-upsampling and feature interpolation, STNet can synthesize high fidelity super-resolution sequence given two low-resolution volumes at both ends as input. Compared to BL, SSR+TSR, and STD, STNet

produces time-varying sequences of better visual quality, both qualitatively and quantitatively. We also compare STNet with TTHRESH to verify its effectiveness.

STNet can be applied to the in-situ scenario: at simulation time, scientists can store the early time steps for STNet training while saving the later time steps sparsely for storage saving. For example, they can keep one time step for every ten time steps simulated and downsize these time steps by 4 at each spatial dimension. During postprocessing, the network is trained with the early time steps only. Once trained, they can recover the super-resolution intermediate time steps with high fidelity, given the sparsely output low-resolution time steps.

CHAPTER 7

CONCLUSIONS AND FUTURE WORKS

7.1 Conclusions

In this dissertation, I have presented several deep learning solutions for scientific data representation and generation. In scientific data representation, first, I proposed an unsupervised representation learning framework for extracting both streamline and stream surface information in high dimensional space. In particular, I developed an encoder decoder structure to learn hidden representations through reconstruction. Further, I established a visual interface to allow users to explore the relationship between the objects and the corresponding hidden representations and understand what information is encoded in the representations. Second, I extended surface representation learning to node representation learning by developing graph convolutional neural network. I studied different unsupervised objective functions and discovered that geodesic distance-based loss can help graph convolutional network learn the most meaningful node representations. I validated the learned representations on node clustering and surface selection tasks and the proposed solution can significantly reduce the training cost compared with convolution-based methods without sacrificing performance. In scientific data generation, I developed a three-stage pipeline for multivariate data selection and translation. Specifically, representation learning is leveraged to estimate the similarity among different variables, translation graph is constructed to determine source and target variables, and generative adversarial network is applied to translate source variable to target variable. This proposed solution can allow domain scientists to partly simulate scientific data while providing a comprehensive in-

sight. Moreover, I proposed an end-to-end generative framework for spatiotemporal super-resolution volume generation. Through sparsely sampled scientific data in both spatial and temporal space, my solution can recover these low-resolution data into high-resolution ones with high quality. Furthermore, inspired by recent pre-training techniques in image classification and segmentation, I designed a pre-training algorithm to improve network generalization during inference stage through cycle loss. This solution can upscale data with 64 or even 512 times in spatial dimension and 3 or even 11 times in temporal dimension, which offers scientists an option to reduce data storage.

7.2 Future works

Although deep learning has achieved impressive results in many fields, such as computer vision and natural language processing, it is not deeply explored in scientific visualization. In the future, there are several open research directions in deep learning for scientific visualization.

Lightweight model. The deep learning models in scientific visualization are built with tens or hundreds of layers to guarantee quality. However, this could result in a large model size and inefficient inference. In the machine learning community, researchers have already studied different techniques (e.g., weight quantization [54, 55, 89] and knowledge distillation [33, 62, 84, 188]) to build a lightweight model from a heavyweight one. For instance, Han et al. [55] pruned the network, quantized parameters and compressed them using Huffman coding. Li et al. [84] applied neural architecture search to find efficient architectures through combining the knowledge of multiple intermediate features extracted from the heavyweight model. The opportunity to incorporate these techniques into deep learning models could significantly improve training efficiency for large-scale scientific data analysis and visualization.

Disentangled learning. The introduced representation learning works are distributed learning, e.g., a network architecture is designed to extract unified features through spe-

cific constraints (e.g., reconstruction). However, the possibility of disentangled learning is still unexplored. In computer vision, Huang et al. [67] built two encoders to learn the content (e.g., shape) and style (e.g., texture and pose) of an image, respectively. After that, fixing the content feature while switching different style features can render arbitrary images with the same content in different styles. Although it is not as intuitive as images to explicitly define content and style for SciVis data. But it still have great potential capability in ensemble data generation. For example, given an ensemble fluid flow simulation, the content could be invariant information (e.g., vortices). The style could be the pattern (e.g., the number, size, and location of vortices) extracted by simulation output under varying Reynolds numbers, which indicate how turbulent the flow is. Following this direction, Disentangled learning can play an important role in controlling and understanding the ensemble simulation process.

Physics-informed. The proposed data generation works for scientific data generation are purely data-driven without considering the underlying physics properties in network training. Recently, researchers in computational fluid dynamics and fluid simulation have extensively investigated physics-informed deep learning solutions [75, 146]. For instance, Raissi et al. [121] introduced a physics-informed neural network for solving supervised learning tasks involving nonlinear partial differential equations. Using differentiable physics and physics-informed deep learning can integrate data and the governing physical laws to produce predictions conforming to the underlying physics.

Federated learning. The success of deep learning models heavily relies on a large amount of data. However, due to practical issues such as confidentiality and privacy, scientific data are often not publicly available. This prevents the deep learning models from gaining a strong learning capability from different sources. Instead of requiring data sets, researchers can release the trained models. Federated learning [74] can produce a shared model by collaborating with local models trained on different data sets. The shared model does not need to access the trained data sets. Techniques in federated learning include

weight averaging [100], momentum update [64], Bayesian non-parametric match [154], and model contrast [85]. The possibilities of designing new approaches when multiple local models are available need to be investigated.

Interpretable DL. In scientific visualization community, researchers treat the deep learning models as black boxes. This makes it difficult to interpret or modify the model results when the predictions are inaccurate or do not meet particular constraints (e.g., the cell size in medical images and physical properties in simulation data). Interpretable deep learning [178] aims to study the role of each neuron in a deep leearning model and understand the decision process. Recent works [7–10] investigated the importance of each neuron in image classification and generation tasks (e.g., identifying the neurons that can control the generation or classification of church). Through this interpretation, researchers can manipulate the model behavior to generate the desired results. For example, by eliminating the sofa neurons, a GAN model can produce images without a sofa. In scientific data generation tasks, we need to identify and discovering the neurons with different roles, controlling specific neurons, and rewriting the neurons to produce customized results when different transfer functions are designed.

Generalization cross tasks. Most of existing deep learning models are tailored for a single task. This design prevents the model learning knowledge from other tasks and generalizing to various tasks. Generalization across tasks aims to design a model that can learn multiple tasks without changing model architecture. In the future, how to build such model could be an interesting and important direction in deep learning for scientific visualization since such a framework can be easily deployed with less effort when domain scientists need to handle different tasks rather than switching to a different deep learning model for each task.

BIBLIOGRAPHY

1. Flow visual. https://sites.nd.edu/chaoli-wang/visvisual/flowvisual/.

2. S. Abu-El-Haija, B. Perozzi, R. Al-Rfou, and A. A. Alemi. Watch your step: Learning node embeddings via graph attention. In *Proceedings of Advances in Neural Information Processing Systems*, pages 9198–9208, 2018.

3. A. Ahmed, N. Shervashidze, S. Narayanamurthy, V. Josifovski, and A. J. Smola. Distributed large-scale natural graph factorization. In *Proceedings of ACM International Conference On World Wide Web*, pages 37–48, 2013.

4. Y. Bai, H. Ding, Y. Qiao, A. Marinovic, K. Gu, T. Chen, Y. Sun, and W. Wang. Unsupervised inductive graph-level representation learning via graph-graph proximity. In *Proceedings of International Joint Conferences on Artificial Intelligence*, pages 1988–1994, 2019.

5. R. Ballester-Ripoll, P. Lindstrom, and R. Pajarola. TTHRESH: Tensor compression for multidimensional visual data. *IEEE Transactions on Visualization and Computer Graphics*, pages 7324–7334, 2019.

6. H. G. Barrow, J. M. Tenenbaum, R. C. Bolles, and H. C. Wolf. Parametric correspondence and chamfer matching: Two new techniques for image matching. In *Proceedings of International Joint Conference on Artificial Intelligence*, pages 659–663, 1977.

7. D. Bau, H. Strobelt, W. Peebles, J. Wulff, B. Zhou, J.-Y. Zhu, and A. Torralba. Semantic photo manipulation with a generative image prior. *ACM Transactions on Graphics*, 38(4):59:1–59:11, 2019.

8. D. Bau, J.-Y. Zhu, H. Strobelt, B. Zhou, J. B. Tenenbaum, W. T. Freeman, and A. Torralba. GAN dissection: Visualizing and understanding generative adversarial networks. In *Proceedings of International Conference on Learning Representations*, 2019.

9. D. Bau, S. Liu, T. Wang, J.-Y. Zhu, and A. Torralba. Rewriting a deep generative model. In *Proceedings of European Conference on Computer Vision*, pages 351–369, 2020.

10. D. Bau, J.-Y. Zhu, H. Strobelt, A. Lapedriza, B. Zhou, and A. Torralba. Understanding the role of individual units in a deep neural network. *Proceedings of the National Academy of Sciences*, 117(48):30071–30078, 2020.

11. Y. Bengio. Learning deep architectures for AI. *Foundations Trends Machine Learning*, 2(1):1–127, 2009.

12. M. Berger, J. Li, and J. A. Levine. A generative model for volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, 25(4):1636–1650, 2019.

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

14. J. M. Blondin and A. Mezzacappa. Pulsar spins from an instability in the accretion shock of supernovae. *Nature*, 445(7123):58–60, 2007.

15. M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst. Geometric deep learning: Going beyond Euclidean data. *Proceedings of IEEE Signal Processing Magazine*, 34(4):18–42, 2017.

16. S. Bruckner and T. Möller. Isosurface similarity maps. *Computer Graphics Forum*, 29(3):773–782, 2010.

17. C.-K. Chen, C. Wang, K.-L. Ma, and A. T. Wittenberg. Static correlation visualization for large time-varying volume data. In *Proceedings of IEEE Pacific Visualization Symposium*, pages 27–34, 2011.

18. T. Chen, S. Kornblith, M. Norouzi, and G. Hinton. A simple framework for contrastive learning of visual representations. In *Proceedings of International Conference on Machine Learning*, 2020.

19. Z. Chen, W. Zeng, Z. Yang, L. Yu, C.-W. Yu, M. Raj, and H. Qu. LassoNet: Deep lasso-selection of 3D point clouds. *IEEE Transactions on Visualization and Computer Graphics*, 26(1):195–204, 2020.

20. H.-C. Cheng, A. Cardone, S. Jain, E. Krokos, K. Narayan, S. Subramaniam, and A. Varshney. Deep-learning-assisted volume visualization. *IEEE Transactions on Visualization and Computer Graphics*, 25(2):1378–1391, 2019.

21. G. Corso, R. Ying, M. Pándy, P. Veličković, J. Leskovec, and P. Liò. Neural distance embeddings for biological sequences. In *Proceedings of Advances in Neural Information Processing Systems*, 2021.

22. R. A. Crawfis and N. Max. Texture splats for 3D scalar and vector field visualization. In *Proceedings of IEEE Visualization*, pages 261–267, 1993.

23. T. Dai, J. Cai, Y. Zhang, S.-T. Xia, and L. Zhang. Second-order attention network for single image super-resolution. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 11065–11074, 2019.

24. C. Doersch, A. Gupta, and A. A. Efros. Unsupervised visual representation learning by context prediction. In *Proceedings of IEEE International Conference on Computer Vision*, pages 1422–1430, 2015.

25. T. Dombre, U. Frisch, J. M. Greene, M. Hénon, A. Mehr, and A. M. Soward. Chaotic streamlines in the ABC flows. *Journal of Fluid Mechanics*, 167:353–391, 1986.

26. J. Donahue and K. Simonyan. Large scale adversarial representation learning. In *Proceedings of Advances in Neural Information Processing Systems*, pages 10542–10552, 2019.

27. C. Dong, C. C. Loy, K. He, and X. Tang. Image super-resolution using deep convolutional networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 38(2):295–307, 2016.

28. M.-L. Eckert, K. Um, and N. Thuerey. ScalarFlow: A large-scale volumetric data set of real-world scalar transport flows for computer animation and machine learning. *ACM Transactions on Graphics*, 38(6):239:1–239:16, 2019.

29. M. Edmunds, R. S. Laramee, R. Malki, I. Masters, N. Croft, G. Chen, and E. Zhang. Automatic stream surface seeding: A feature centered approach. *Computer Graphics Forum*, 31(3):1095–1104, 2012.

30. D. Erhan, A. Courville, Y. Bengio, and P. Vincent. Why does unsupervised pretraining help deep learning? In *Proceedings of International Conference on Artificial Intelligence and Statistics*, pages 201–208, 2010.

31. M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 226–231, 1996.

32. J. M. Esturo, M. Schulze, C. Rössl, and H. Theisel. Global selection of stream surfaces. *Computer Graphics Forum*, 32(2):113–122, 2013.

33. Z. Fang, J. Wang, X. Hu, L. Wang, Y. Yang, and Z. Liu. Compressing visual-linguistic model via knowledge distillation. In *Proceedings of IEEE International Conference on Computer Vision*, pages 1428–1438, 2021.

34. M. Garland and P. S. Heckbert. Surface simplification using quadric error metrics. In *Proceedings of ACM SIGGRAPH Conference*, pages 209–216, 1997.

35. S. Gidaris, P. Singh, and N. Komodakis. Unsupervised representation learning by predicting image rotations. In *Proceedings of International Conference on Learning Representations*, 2018.

36. R. Girdhar, D. F. Fouhey, M. Rodriguez, and A. Gupta. Learning a predictable and generative vector representation for objects. In *Proceedings of European Conference on Computer Vision*, pages 484–499, 2016.

37. M. Glatter, C. Mollenhour, J. Huang, and J. Gao. Scalable data servers for large multivariate volume visualization. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):1291–1299, 2006.

38. P. Golik, P. Doetsch, and H. Ney. Cross-entropy vs. squared error training: A theoretical and experimental comparison. In *Proceedings of Conference of the International Speech Communication Association*, pages 1756–1760, 2013.

39. I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016.

40. J.-B. Grill, F. Strub, F. Altché, C. Tallec, P. Richemond, E. Buchatskaya, C. Doersch, B. Avila Pires, Z. Guo, M. Gheshlaghi Azar, B. Piot, K. Kavukcuoglu, R. Munos, and M. Valko. Bootstrap your own latent: A new approach to self-supervised learning. In *Proceedings of Advances in Neural Information Processing Systems*, 2020.

41. P. Gu, J. Han, D. Z. Chen, and C. Wang. Reconstructing unsteady flow data from representative streamlines via diffusion and deep-learning-based denoising. *IEEE Computer Graphics and Applications*, 41(6):111–121, 2021.

42. P. Gu, J. Han, D. Z. Chen, and C. Wang. Scalar2Vec: Translating scalar fields to vector fields via deep learning. In *Proceedings of IEEE Pacific Visualization Symposium*, 2022. Accepted.

43. 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.

44. W. Hamilton, Z. Ying, and J. Leskovec. Inductive representation learning on large graphs. In *Proceedings of Advances in Neural Information Processing Systems*, pages 1024–1034, 2017.

45. 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. Accepted.

46. 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.

47. J. Han and C. Wang. TSR-VFD: Generating temporal super- resolution for unsteady vector field data. *Computers & Graphics*, 2022. Accepted.

48. J. Han and C. Wang. SurfNet: Learning surface representations via graph convolutional network. *Computer Graphics Forum*, 2022. Conditionally Accepted.

49. J. Han and C. Wang. VCNet: A generative model for volume completion. *Visual Informatics*, 2022. Conditionally Accepted.

50. 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.

51. 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.

52. J. Han, H. Zheng, Y. Xing, D. Z. Chen, and C. Wang. V2V: A deep learning approach to variable-to-variable selection and translation for multivariate time-varying data. *IEEE Transactions on Visualization and Computer Graphics*, 27(2):1290–1300, 2021.

53. J. Han, H. Zheng, D. Z. Chen, and C. Wang. STNet: An end-to-end generative framework for synthesizing spatiotemporal super-resolution volumes. *IEEE Transactions on Visualization and Computer Graphics*, 28(1):270–280, 2022.

54. S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. Horowitz, and B. Dally. Deep compression and EIE: Efficient inference engine on compressed deep neural network. In *Proceedings of IEEE Hot Chips Symposium*, pages 1–6, 2016.

55. S. Han, H. Mao, and W. J. Dally. Deep compression: Compressing deep neural network with pruning, trained quantization and Huffman coding. In *Proceedings of International Conference on Learning Representations*, 2016.

56. R. Hanocka, A. Hertz, N. Fish, R. Giryes, S. Fleishman, and D. Cohen-Or. MeshCNN: A network with an edge. *ACM Transactions on Graphics*, 38(4):90:1–90:12, 2019.

57. K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of IEEE International Conference on Computer Vision*, pages 1026–1034, 2015.

58. K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

59. K. He, H. Fan, Y. Wu, S. Xie, and R. Girshick. Momentum contrast for unsupervised visual representation learning. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 9729–9738, 2020.

60. W. He, J. Wang, H. Guo, K.-C. Wang, H.-W. Shen, M. Raj, Y. S. G. Nashed, and T. Peterka. InSituNet: Deep image synthesis for parameter space exploration of ensemble simulations. *IEEE Transactions on Visualization and Computer Graphics*, 26(1):23–33, 2020.

61. D. Hendrycks, K. Lee, and M. Mazeika. Using pre-training can improve model robustness and uncertainty. In *Proceedings of International Conference for Learning Representations*, 2019.

62. G. E. Hinton, O. Vinyals, and J. Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

63. 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.

64. T.-M. H. Hsu, H. Qi, and M. Brown. Measuring the effects of non-identical data distribution for federated visual classification. *arXiv preprint arXiv:1909.06335*, 2019.

65. G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger. Densely connected convolutional networks. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 4700–4708, 2017.

66. J. Huang, Z. Li, N. Li, S. Liu, and G. Li. AttPool: Towards hierarchical feature representation in graph convolutional networks via attention mechanism. In *Proceedings of IEEE International Conference on Computer Vision*, pages 6480–6489, 2019.

67. X. Huang, M.-Y. Liu, S. Belongie, and J. Kautz. Multimodal unsupervised image-to-image translation. In *Proceedings of European Conference on Computer Vision*, pages 172–189, 2018.

68. S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of International Conference on Machine Learning*, pages 448–456, 2015.

69. P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros. Image-to-image translation with conditional adversarial networks. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 1125–1134, 2017.

70. 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.

71. B. Jiang, Z. Zhang, D. Lin, J. Tang, and B. Luo. Semi-supervised learning with graph learning-convolutional networks. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 11313–11320, 2019.

72. H. Jiang, D. Sun, V. Jampani, M.-H. Yang, E. Learned-Miller, and J. Kautz. Super SloMo: High quality estimation of multiple intermediate frames for video interpolation. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 9000–9008, 2018.

73. Y. Jo, S. W. Oh, J. Kang, and S. J. Kim. Deep video super-resolution network using dynamic upsampling filters without explicit motion compensation. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 3224–3232, 2018.

74. P. Kairouz, H. B. McMahan, B. Avent, A. Bellet, M. Bennis, A. N. Bhagoji, K. A. Bonawitz, Z. Charles, G. Cormode, R. Cummings, R. G. L. D'Oliveira, H. Eichner, S. E. Rouayheb, D. Evans, J. Gardner, Z. Garrett, A. Gascón, B. Ghazi, P. B. Gibbons,

M. Gruteser, Z. Harchaoui, C. He, L. He, Z. Huo, B. Hutchinson, J. Hsu, M. Jaggi, T. Javidi, G. Joshi, M. Khodak, J. Konečný, A. Korolova, F. Koushanfar, S. Koyejo, T. Lepoint, Y. Liu, P. Mittal, M. Mohri, R. Nock, A. Özgür, R. Pagh, H. Qi, D. Ramage, R. Raskar, M. Raykova, D. Song, W. Song, S. U. Stich, Z. Sun, A. T. Suresh, F. Tramèr, P. Vepakomma, J. Wang, L. Xiong, Z. Xu, Q. Yang, F. X. Yu, H. Yu, and S. Zhao. Advances and open problems in federated learning. *Foundations and Trends in Machine Learning*, 14(1-2):1–210, 2021.

75. G. E. Karniadakis, I. G. Kevrekidis, L. Lu, P. Perdikaris, S. Wang, and L. Yang. Physics-informed machine learning. *Nature Reviews Physics*, 3:422–440, 2021.

76. 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.

77. B. Kim, V. C. Azevedo, N. Thuerey, T. Kim, M. Gross, and B. Solenthaler. Deep Fluids: A generative network for parameterized fluid simulations. *Computer Graphics Forum*, 38(2):59–70, 2019.

78. D. Kingma and J. Ba. Adam: A method for stochastic optimization. In *Proceedings of International Conference for Learning Representations*, 2015.

79. T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. In *Proceedings of International Conference for Learning Representations*, 2017.

80. I. Kostrikov, Z. Jiang, D. Panozzo, D. Zorin, and J. Bruna. Surface networks. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 2540–2548, 2018.

81. J. B. Kruskal. Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis. *Psychometrika*, 29(1):1–27, 1964.

82. G. Larsson, M. Maire, and G. Shakhnarovich. Learning representations for automatic colorization. In *Proceedings of the European Conference on Computer Vision*, pages 577–593, 2016.

83. C. Ledig, L. Theis, F. Huszár, J. Caballero, A. Cunningham, A. Acosta, A. P. Aitken, A. Tejani, J. Totz, Z. Wang, and W. Shi. Photo-realistic single image super-resolution using a generative adversarial network. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 4681–4690, 2017.

84. M. Li, J. Lin, Y. Ding, Z. Liu, J.-Y. Zhu, and S. Han. GAN compression: Efficient architectures for interactive conditional GANs. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 5284–5294, 2020.

85. Q. Li, B. He, and D. Song. Model-contrastive federated learning. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 10713–10722, 2021.

86. X. Liang, S. Di, D. Tao, Z. Chen, and F. Cappello. An efficient transformation scheme for lossy data compression with point-wise relative error bound. In *Proceedings of IEEE International Conference on Cluster Computing*, pages 179–189, 2018.

87. 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.

88. J. Lin. Divergence measures based on the Shannon entropy. *IEEE Transactions on Information theory*, 37(1):145–151, 1991.

89. J. Lin, C. Gan, and S. Han. Defensive quantization: When efficiency meets robustness. In *Proceedings of International Conference on Learning Representations*, 2019.

90. M. Lin, Q. Chen, and S. Yan. Network in network. In *Proceedings of International Conference for Learning Representations*, 2014.

91. 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.

92. O. Litany, A. Bronstein, M. Bronstein, and A. Makadia. Deformable shape completion with graph convolutional autoencoders. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 1886–1895, 2018.

93. C. Liu, H. Ji, and A. Qiu. Convolutional neural network on semi-regular triangulated meshes and its application to brain image data. *arXiv preprint arXiv:1903.08828*, 2019.

94. X. Liu and H.-W. Shen. Association analysis for visual exploration of multivariate scientific data sets. *IEEE Transactions on Visualization and Computer Graphics*, 22 (1):955–964, 2016.

95. Z. Liu, R. A. Yeh, X. Tang, Y. Liu, and A. Agarwala. Video frame synthesis using deep voxel flow. In *Proceedings of IEEE International Conference on Computer Vision*, pages 4463–4471, 2017.

96. K. Lu, A. Chaudhuri, T.-Y. Lee, H.-W. Shen, and P. C. Wong. Exploring vector fields with distribution-based streamline analysis. In *Proceedings of IEEE Pacific Visualization Symposium*, pages 257–264, 2013.

97. J. Ma, C. Wang, and C.-K. Shene. Coherent view-dependent streamline selection for importance-driven flow visualization. In *IS&T/SPIE Visualization and Data Analysis*, pages 865407–1–865407–15, 2013.

98. K.-L. Ma. Machine learning to boost the next generation of visualization technology. *IEEE Computer Graphics and Applications*, 27(5):6–9, 2007.

99. S. Marchesin, C.-K. Chen, C. Ho, and K.-L. Ma. View-dependent streamlines for 3D vector fields. *IEEE Transactions on Visualization and Computer Graphics*, 16(6): 1578–1586, 2010.

100. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Proceedings of International Conference on Artificial Intelligence and Statistics*, pages 1273–1282, 2017.

101. S. Menon, A. Damian, S. Hu, N. Ravi, and C. Rudin. PULSE: Self-supervised photo upsampling via latent space exploration of generative models. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 2437–2445, 2020.

102. S. Meyer, O. Wang, H. Zimmer, M. Grosse, and A. Sorkine-Hornung. Phase-based frame interpolation for video. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 1410–1418, 2015.

103. F. Milletari, N. Navab, and S.-A. Ahmadi. V-Net: Fully convolutional neural networks for volumetric medical image segmentation. In *Proceedings of IEEE International Conference on 3D vision*, pages 565–571, 2016.

104. T. Miyato, T. Kataoka, M. Koyama, and Y. Yoshida. Spectral normalization for generative adversarial networks. In *Proceedings of International Conference for Learning Representations*, 2018.

105. F. Monti, D. Boscaini, J. Masci, E. Rodola, J. Svoboda, and M. M. Bronstein. Geometric deep learning on graphs and manifolds using mixture model CNNs. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 5115–5124, 2017.

106. U. Mudenagudi, S. Banerjee, and P. K. Kalra. Space-time super-resolution using graph-cut optimization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(5):995–1008, 2010.

107. V. Nair and G. E. Hinton. Rectified linear units improve restricted Boltzmann machines. In *Proceedings of International Conference on Machine Learning*, pages 807–814, 2010.

108. A.-D. Nguyen, W. Kim, J. Kim, and S. Lee. Video frame interpolation by plug-and-play deep locally linear embedding. *arXiv preprint arXiv:1807.01462*, 2018.

109. S. Niklaus, L. Mai, and F. Liu. Video frame interpolation via adaptive convolution. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 670–679, 2017.

110. B. Niu, W. Wen, W. Ren, X. Zhang, L. Yang, S. Wang, K. Zhang, X. Cao, and H. Shen. Single image super-resolution via a holistic attention network. In *Proceedings of European Conference on Computer Vision*, pages 191–207, 2020.

111. S. Oeltze, D. J. Lehmann, A. Kuhn, G. Janiga, H. Theisel, and B. Preim. Blood flow clustering and applications in virtual stenting of intracranial aneurysms. *IEEE Transactions on Visualization and Computer Graphics*, 20(5):686–701, 2014.

112. T. Park, M.-Y. Liu, T.-C. Wang, and J.-Y. Zhu. Semantic image synthesis with spatially-adaptive normalization. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 2337–2346, 2019.

113. A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. PyTorch: An imperative style, high-performance deep learning library. In *Proceedings of Advances in Neural Information Processing Systems*, pages 8024–8035, 2019.

114. D. Pathak, P. Krahenbuhl, J. Donahue, T. Darrell, and A. A. Efros. Context encoders: Feature learning by inpainting. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 2536–2544, 2016.

115. E. Pérez-Pellitero, M. S. Sajjadi, M. Hirsch, and B. Schölkopf. Photorealistic video super resolution. *arXiv preprint arXiv:1807.07930*, 2018.

116. S. Popinet, M. Smith, and C. Stevens. Experimental and numerical study of the turbulence characteristics of airflow around a research vessel. *Journal of Atmospheric and Oceanic Technology*, 21(10):1575–1589, 2004.

117. W. P. Porter, Y. Xing, B. R. von Ohlen, J. Han, and C. Wang. A deep learning approach to selecting representative time steps for time-varying multivariate data. In *Proceedings of IEEE Conference on Visualization (Short Papers)*, pages 131–135, 2019.

118. L. Prantl, B. Bonev, and N. Thuerey. Generating liquid simulations with deformation-aware neural networks. In *Proceedings of International Conference for Learning Representations*, 2019.

119. C. R. Qi, H. Su, M. Nießner, A. Dai, M. Yan, and L. J. Guibas. Volumetric and multi-view cnns for object classification on 3D data. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 5648–5656, 2016.

120. H. Qu, W.-Y. Chan, A. Xu, K.-L. Chung, K.-H. Lau, and P. Guo. Visual analysis of the air pollution problem in Hong Kong. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1408–1415, 2007.

121. M. Raissi, P. Perdikaris, and G. E. Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving non-linear partial differential equations. *Journal of Computational Physics*, 378:686–707, 2019.

122. M. Raji, A. Hota, R. Sisneros, P. Messmer, and J. Huang. Photo-guided exploration of volume data features. In *Proceedings of Eurographics Symposium on Parallel Graphics and Visualization*, pages 31–39, 2017.

123. A. Ranjan, T. Bolkart, S. Sanyal, and M. J. Black. Generating 3D faces using convolutional mesh autoencoders. In *Proceedings of European Conference on Computer Vision*, pages 704–720, 2018.

124. M. P. Rast. The scales of granulation, mesogranulation, and supergranulation. *The Astrophysical Journal*, 597(2):1200–1210, 2003.

125. I. B. Rojo and T. Günther. Vector field topology of time-dependent flows in a steady reference frame. *IEEE Transactions on Visualization and Computer Graphics*, 26(1): 280–290, 2019.

126. O. Ronneberger, P. Fischer, and T. Brox. U-Net: Convolutional networks for biomedical image segmentation. In *Proceedings of International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 234–241, 2015.

127. C. Rössl and H. Theisel. Streamline embedding for 3D vector field exploration. *IEEE Transactions on Visualization and Computer Graphics*, 18(3):407–420, 2012.

128. K. Roth, A. Lucchi, S. Nowozin, and T. Hofmann. Stabilizing training of generative adversarial networks through regularization. In *Proceedings of Advances in Neural Information Processing Systems*, pages 2018–2028, 2017.

129. C. Rother, T. Minka, A. Blake, and V. Kolmogorov. Cosegmentation of image pairs by histogram matching-incorporating a global constraint into MRFs. In *Proceedings of IEEE International Conference on Computer Vision*, pages 993–1000, 2006.

130. M. S. Sajjadi, R. Vemulapalli, and M. Brown. Frame-recurrent video super-resolution. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 6626–6634, 2018.

131. N. Sauber, H. Theisel, and H.-P. Seidel. Multifield-Graphs: An approach to visualizing correlations in multifield scalar data. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):917–924, 2006.

132. M. Schulze, J. Martinez Esturo, T. Günther, C. Rössl, H.-P. Seidel, T. Weinkauf, and H. Theisel. Sets of globally optimal stream surfaces for flow visualization. *Computer Graphics Forum*, 33(3):1–10, 2014.

133. O. Shahar, A. Faktor, and M. Irani. Space-time super-resolution from a single video. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 3353–3360, 2011.

134. A. Shamir. A survey on mesh segmentation techniques. *Computer Graphics Forum*, 27(6):1539–1556, 2008.

135. E. Shechtman, Y. Caspi, and M. Irani. Increasing space-time resolution in video. In *Proceedings of European Conference on Computer Vision*, pages 753–768, 2002.

136. N. Shi and Y. Tao. CNNs based viewpoint estimation for volume visualization. *ACM Transactions on Intelligent Systems and Technology*, 10(3):27:1–27:22, 2019.

137. Z. Shu, C. Qi, S. Xin, C. Hu, L. Wang, Y. Zhang, and L. Liu. Unsupervised 3D shape segmentation and co-segmentation via deep learning. *Computer Aided Geometric Design*, 43:39–52, 2016.

138. E. Smith, S. Fujimoto, A. Romero, and D. Meger. GEOMetrics: Exploiting geometric structure for graph-encoded objects. In *Proceedings of International Conference on Machine Learning*, volume 97, pages 5866–5876, 2019.

139. H. Su, S. Maji, E. Kalogerakis, and E. Learned-Miller. Multi-view convolutional neural networks for 3d shape recognition. In *Proceedings of IEEE International Conference on Computer Vision*, pages 945–953, 2015.

140. J. Sukharev, C. Wang, K.-L. Ma, and A. T. Wittenberg. Correlation study of time-varying multivariate climate data sets. In *Proceedings of IEEE Pacific Visualization Symposium*, pages 161–168, 2009.

141. H. Takeda, P. Van Beek, and P. Milanfar. Spatiotemporal video upscaling using motion-assisted steering kernel (mask) regression. In M. Mrak, M. Grgic, and M. Kunt, editors, *High-Quality Visual Experience*, pages 245–274. 2010.

142. J. Tao and C. Wang. Semi-automatic generation of stream surfaces via sketching. *IEEE Transactions on Visualization and Computer Graphics*, 24(9):2622–2635, 2018.

143. 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.

144. J. Tao, M. Imre, C. Wang, N. V. Chawla, H. Guo, G. Sever, and S. H. Kim. Exploring time-varying multivariate volume data using matrix of isosurface similarity maps. *IEEE Transactions on Visualization and Computer Graphics*, 25(1):1236–1245, 2019.

145. J. B. Tenenbaum, V. de Silva, and J. C. Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290(5500):2319–2323, 2000.

146. N. Thuerey, P. Holl, M. Mueller, P. Schnell, F. Trost, and K. Um. Physics-based deep learning. *arXiv preprint arXiv:2109.05237*, 2021.

147. G. Tkachev, S. Frey, and T. Ertl. Local prediction models for spatiotemporal volume visualization. *IEEE Transactions on Visualization and Computer Graphics*, 2019. Accepted.

148. G. Tkachev, S. Frey, and T. Ertl. S4: Self-supervised learning of spatiotemporal similarity. *IEEE Transactions on Visualization and Computer Graphics*, 2021. Accepted.

149. F.-Y. Tzeng, E. B. Lum, and K.-L. Ma. A novel interface for higher-dimensional classification of volume data. In *Proceedings of IEEE Visualization Conference*, pages 505–512, 2003.

150. F.-Y. Tzeng, E. B. Lum, and K.-L. Ma. An intelligent system approach to higher-dimensional classification of volume data. *IEEE Transactions on Visualization and Computer Graphics*, 11(3):273–284, 2005.

151. L. van der Maaten and G. Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9(11):2579–2605, 2008.

152. W. von Funck, T. Weinkauf, H. Theisel, and H.-P. Seidel. Smoke surfaces: An interactive flow visualization technique inspired by real-world flow experiments. *IEEE Transactions on Visualization and Computer Graphics*, 14(6):1396–1403, 2008.

153. C. Wang, H. Yu, R. W. Grout, K.-L. Ma, and J. H. Chen. Analyzing information transfer in time-varying multivariate data. In *Proceedings of IEEE Pacific Visualization Symposium*, pages 99–106, 2011.

154. H. Wang, M. Yurochkin, Y. Sun, D. Papailiopoulos, and Y. Khazaeni. Federated learning with matched averaging. In *Proceedings of International Conference on Learning Representations*, 2020.

155. N. Wang, Y. Zhang, Z. Li, Y. Fu, W. Liu, and Y.-G. Jiang. Pixel2mesh: Generating 3D mesh models from single RGB images. In *Proceedings of European Conference on Computer Vision*, pages 52–67, 2018.

156. T.-C. Wang, M.-Y. Liu, A. Tao, G. Liu, J. Kautz, and B. Catanzaro. Few-shot video-to-video synthesis. In *Proceedings of Advances in Neural Information Processing Systems*, 2019.

157. X. Wang, K. Yu, S. Wu, J. Gu, Y. Liu, C. Dong, Y. Qiao, and C. Change Loy. ESGAN: Enhanced super-resolution generative adversarial networks. In *Proceedings of European Conference on Computer Vision Workshops*, 2018.

158. Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, and J. M. Solomon. Dynamic graph CNN for learning on point clouds. *ACM Transactions on Graphics*, 38(5): 146:1–146:12, 2019.

159. Z. Wang, J. Chen, and S. C. H. Hoi. Deep learning for image super-resolution: A survey. *arXiv preprint arXiv:1902.06068*, 2019.

160. J. Wei, C. Wang, H. Yu, and K.-L. Ma. A sketch-based interface for classifying and visualizing vector fields. In *Proceedings of IEEE Pacific Visualization Symposium*, pages 129–136, 2010.

161. D. Weiskopf, T. Schafhitzel, and T. Ertl. Real-time advection and volumetric illumination for the visualization of 3D unsteady flow. In *Proceedings of Eurographics / IEEE VGTC Symposium on Visualization*, pages 13–20, 2005.

162. S. Weiss, M. Chu, N. Thuerey, and R. Westermann. Volumetric isosurface rendering with deep learning-based super-resolution. *IEEE Transactions on Visualization and Computer Graphics*, 27(6):3064–3078, 2021.

163. M. Werhahn, Y. Xie, M. Chu, and N. Thuerey. A multi-pass GAN for fluid flow super-resolution. *Proceedings of the ACM on Computer Graphics and Interactive Techniques*, 2(2):1–21, 2019.

164. 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.

165. Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao. 3D ShapeNets: A deep representation for volumetric shapes. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 1912–1920, 2015.

166. X. Xiang, Y. Tian, Y. Zhang, Y. Fu, J. P. Allebach, and C. Xu. Zooming Slow-Mo: Fast and accurate one-stage space-time video super-resolution. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 3370–3379, 2020.

167. 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.

168. 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.

169. K. Xu, M. Zhang, S. Jegelka, and K. Kawaguchi. Optimization of graph neural networks: Implicit acceleration by skip connections and more depth. In *Proceedings of International Conference on Machine Learning*, pages 11592–11602, 2021.

170. 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.

171. G. Yao, Y. Yuan, T. Shao, and K. Zhou. Mesh guided one-shot face reenactment using graph convolutional networks. In *Proceedings of ACM International Conference on Multimedia*, pages 1773–1781, 2020.

172. X. Ye, D. Kao, and A. Pang. Strategy for seeding 3D streamlines. In *Proceedings of IEEE Visualization Conference*, pages 471–478, 2005.

173. L. Yi, H. Su, X. Guo, and L. J. Guibas. SyncSpecCNN: Synchronized spectral cnn for 3d shape segmentation. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 2282–2290, 2017.

174. R. Ying, R. He, K. Chen, P. Eksombatchai, W. L. Hamilton, and J. Leskovec. Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 974–983, 2018.

175. Z. Ying, J. You, C. Morris, X. Ren, W. Hamilton, and J. Leskovec. Hierarchical graph representation learning with differentiable pooling. In *Proceedings of Advances in Neural Information Processing Systems*, pages 4800–4810, 2018.

176. H. Yu, C. Wang, C.-K. Shene, and J. H. Chen. Hierarchical streamline bundles. *IEEE Transactions on Visualization and Computer Graphics*, 18(8):1353–1367, 2012.

177. Y. Yuan, S. Liu, J. Zhang, Y. Zhang, C. Dong, and L. Lin. Unsupervised image super-resolution using cycle-in-cycle generative adversarial networks. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 701–710, 2018.

178. Q. Zhang and S.-C. Zhu. Visual interpretability for deep learning: A survey. *Frontiers of Information Technology & Electronic Engineering*, 19(1):27–39, 2018.

179. R. Zhang. Making convolutional networks shift-invariant again. In *Proceedings of IEEE International Conference on Machine Learning*, pages 7324–7334, 2019.

180. R. Zhang, P. Isola, and A. A. Efros. Colorful image colorization. In *Proceedings of European Conference on Computer Vision*, pages 649–666, 2016.

181. R. Zhang, J.-Y. Zhu, P. Isola, X. Geng, A. S. Lin, T. Yu, and A. A. Efros. Real-time user-guided image colorization with learned deep priors. *ACM Transactions on Graphics*, 9(4):119:1–119:11, 2017.

182. R. Zhang, P. Isola, A. A. Efros, E. Shechtman, and O. Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 586–595, 2018.

183. Y. Zhang, K. Li, K. Li, L. Wang, B. Zhong, and Y. Fu. Image super-resolution using very deep residual channel attention networks. In *Proceedings of European Conference on Computer Vision*, pages 294–310, 2018.

184. H. Zheng, L. Yang, J. Chen, J. Han, Y. Zhang, P. Liang, Z. Zhao, C. Wang, and D. Z. Chen. Biomedical image segmentation via representative annotation. In *Proceedings of AAAI Conference on Artificial Intelligence*, pages 5901–5908, 2019.

185. H. Zheng, S. M. M. Perrine, M. K. Pitirri, K. Kawasaki, C. Wang, J. T. Richtsmeier, and D. Z. Chen. Cartilage segmentation in high-resolution 3d micro-ct images via

uncertainty-guided self-training with very sparse annotation. In *Proceedings of International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 802–812, 2020.

186. Z. Zhou, Y. Hou, Q. Wang, G. Chen, J. Lu, Y. Tao, and H. Lin. Volume upscaling with convolutional neural networks. In *Proceedings of Computer Graphics International*, pages 38:1–38:6, 2017.

187. J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Proceedings of IEEE International Conference on Computer Vision*, pages 2223–2232, 2017.

188. Y. Zhu and Y. Wang. Student customized knowledge distillation: Bridging the gap between student and teacher. In *Proceedings of IEEE International Conference on Computer Vision*, pages 5057–5066, 2021.