Time Varying Particle Data Feature Extraction and Tracking with Neural Networks

Category: Research

Paper Type: Data Transformations

Abstract—Analyzing particle data plays an important role in many scientific applications such as fluid simulation, cosmology simulation and molecular dynamics. While there exist methods that can perform feature extraction and tracking for volumetric data, performing those tasks for particle data is more challenging because of the lack of explicit connectivity information. Although one may convert the particle data to volume first, this approach is at risk of incurring error and increasing the size of the data. In this paper, we take a deep learning approach to create feature representations for scientific particle data to assist feature extraction and tracking. We employ a deep learning model, which produces latent vectors to represent the relation between spatial locations and physical attributes in a local neighborhood. With the latent vectors, features can be extracted by clustering these vectors. To achieve fast feature tracking, the mean-shift tracking algorithm is applied in the feature space, which only requires inference of the latent vector for selected regions of interest. We validate our approach using two datasets and compare our method with other existing methods.

Index Terms—Feature extraction and tracking, Time-varying data analysis, Particle data, Deep learning,

1 Introduction

With the growing size and complexity of simulations, feature extraction and tracking have become an essential task for effective scientific data analysis. Through the extraction and tracking of features, scientists can better understand the evolution of scientific phenomena, and overcome the difficulties in visualizing large scale data such as 3D occlusion and expensive I/O because they only need to focus on features at reduced complexity and sizes instead of the whole dataset. From the early work [33] to the more recent approaches such as topology analysis [25] and deep learning [27], feature extraction and tracking methods generally share a common strategy that is to find the spatially unique and temporally coherent regions based on some feature specific attributes such as values, size, shape, topology or texture.

When it comes to particle data, such as those produced by molecular dynamics, cosmology, or fluid flow simulations, feature extraction and tracking still face many challenges. The lack of connectivity between particles makes it difficult to define features based on gradients or textures, or directly apply methods such as topology analysis and region growing. One common solution is to re-sample the attributes of particles into a regular grid and then perform feature extraction and tracking [11, 25]. However, re-sampling data may cause loss of details or significant increase of data storage size. While it is true that particles produced by pure Lagrangian methods are uniquely indexed throughout the whole time sequence, and hence tracking particles can be more easily done, particle tracking does not equate to feature tracking. A feature may evolve over time and hence the set of particles that define the feature will evolve as well. In this case, while feature tracking can be assisted by tracking the movement of corresponding particles, it is still necessary to perform feature tracking to identify the dynamic set of feature particles in all time steps [35]. Moreover, there are situations when particle indices are not available [12] or discarded to reduce the storage size.

Besides the inherent challenges for particles data, finding a precise definition for salient features can be difficult due to the complexity of many scientific phenomena. Many features cannot be separated by simple thresholds of attribute values, and extracting features with hand-crafted feature descriptors is tedious and not always robust. To solve this problem, deep learning has been used to automatically extract feature descriptors based on the knowledge of domain experts [7] or feature statistics [27]. The aforementioned two approaches, however, are not appropriate for "soft-knowledge features", which are defined by Luciani et al [24] as features that are difficult to be captured and represented computationally, such as viscous fingers in the particle data from 2016 SciVis Contest. Providing feature labels by domain experts as training data, as is done in [7], is time consuming and difficult without a method to solve the problems of efficient visualization first.

To tackle the aforementioned two major challenges (1) the missing of connectivity in particle data, (2) difficulty in utilizing "soft-knowledge" about features, in this paper, we propose a novel neural network based approach. One particle and its local neighborhood, hereafter we call a particle patch through out this paper, is processed by neural networks to produce a latent vector without needing the connectivity explicitly. Feature extraction is then achieved by grouping particles of similar latent vectors into clusters. Our network model is based upon a state-of-the-art architecture [21] for point cloud data input with a modification so that we can enforce that the features learnt by the model are the relation between particle attributes and their spatial location in the local neighborhood. Our model do not need any feature labels in the training data, and instead an unsupervised training scheme called autoencoder is adopted. The domain expert's knowledge is injected by exploring and choosing the plausible features from clusters of latent vectors. Finally, chosen features are tracked using the mean-shift tracking algorithm based on the distributions of latent vectors. The proposed approach is applicable to noisy and multi-variate datasets, which poses additional challenges in feature extraction.

We summarize the contribution of this study as follows:

- Feature extraction with a novel neural network architecture specifically designed for particle data that can capture the spatial-physical attribute relation in the local neighborhood without explicit spatial connectivity.
- Fast feature tracking approach that exploits mean-shift tracking using latent distributions.
- A demonstration of applying our method to successfully perform feature extraction and tracking for two noisy and multi-variate particle datasets.

The remaining of this paper is organized as follows: In section 2, we present the related literature to this study. In section 3-5, we explain our proposed approach from an overview to the details. In Section 6 we evaluate the approach using two publicly available particle datasets. Section 7 contains the discussion and the future work of this study.

2 Related Works

Particle simulations are used across a plethora of scientific fields. Analyzing and understanding this kind of data is a hot topic, as evidenced by the fact that 2015, 2016 and 2019 Scientific Visualization Contests all focused on particle datasets [8, 12, 36]. For each of these datasets,

the tasks of feature extraction and tracking were critical for understanding the development of the simulated phenomena over time. Recent work [32] also proposed a method to interactively visualize large scale particle datasets by representing them with Gaussian Mixture Models (GMMs).

The approach proposed in this paper is inspired by the deep learning assisted volume visualization by Cheng et al. [7], where supervised deep learning neural networks are used as a tool to extract meaningful features to help the design of a good transfer function for volume rendering. Another related recent study by Han el al [14] uses the proximity in the latent space to select the best pairs of variables to perform volumetric data transformation. Meanwhile, recent works in the deep learning domain have designed and applied neural networks [15, 30, 31, 40] to point cloud data and have seen great success for classification and segmentation, where both require meaningful feature extraction from point clouds. Since successful classification and segmentation indicates successful feature learning from the dataset, we survey former studies in both the fields of feature extraction and tracking and point cloud neural networks. Moreover, in this paper, the mean-shift tracking algorithm from the domain of computer vision is used. Therefore we also give a brief survey of this technique.

2.1 Feature Extraction and Tracking

Most feature extraction and tracking techniques fall into two categories. The first and the most prevalent approaches [5, 10, 19, 28, 33, 37, 43] perform extraction and tracking separately. Features are either extracted by spatially coherent attributes, or extracted by domain specific handcrafted feature descriptors. After that, tracking is done by matching the features using spatial overlap over time. The Second method [20,34,41] consider features directly in the spatial-temporal space and extract them by topology analysis or high-dimensional iso-surfacing. It's worth to mention that machine learning techniques have also been used for feature extraction and tracking. An early work by Tzeng and Ma [39] proposed an approach to extract features and design transfer functions using Machine Learning for flow field data. Morfort et al [27] showed that deep learning can be used to detect shock locations in turbulent combustion tensor fields. However, most of the feature extraction methods cannot be directly applied to particle data due to the lack of connectivity information among the particles.

There have been some works aiming to solve this problem for particle datasets. Silver and Wang [37] extended their former volume feature extraction and tracking algorithm to unstructured datasets by using a new data structure instead of octrees. Lukasczyk et al [25] resampled irregular volumetric points into a regular volume before topology analysis is applied to the data set. Similarly, the 2016 SciVis contest winners [12] proposed an approach which constructs a smooth scalar field before using a contour tree to extract critical points. Linsen et al [23] used a region growing method in the attribute space to solve the problem of extracting features defined over multiple variables in a particle data set. Work by Dutta and Shen [10] utilized an incremental Gaussian Mixture Model to both detect and track features. Although their experiments are done with volumetric datasets, there method can be adopted to scattered data. However these works either cannot extract "soft" or "fuzzy" features or does not fully demonstrate its effectiveness on particle data.

2.2 Point Cloud Neural Networks and their Visualization Application

A deep learning survey for 3D point clouds provides a comprehensive overview of the recent point cloud neural network studies [13]. The PointNet [30] is one of the first methods in deep learning for point cloud data. Subsequently, Qi et al. expended the PointNet with PointNet++ by applying the PointNet recursively on a nested partitioning of the input point cloud to improve efficiency and robustness [31]. PointNet++ borrows the ideas of hierarchically applying filters to extract both local and global features from Convolutional Neural Networks(CNN). Lots of recent works improve on the basic architecture of PointNet++ [15, 17, 21, 22, 38, 40, 42]. In our work, GeoConv [21] is adopted in our own neural networks as it shows good performance with relative small network size. LassoNet [6], which solves the problem of interactive selection of objects with lasso is one of the few visualization works we can find that utilizes 3D point cloud neural networks.

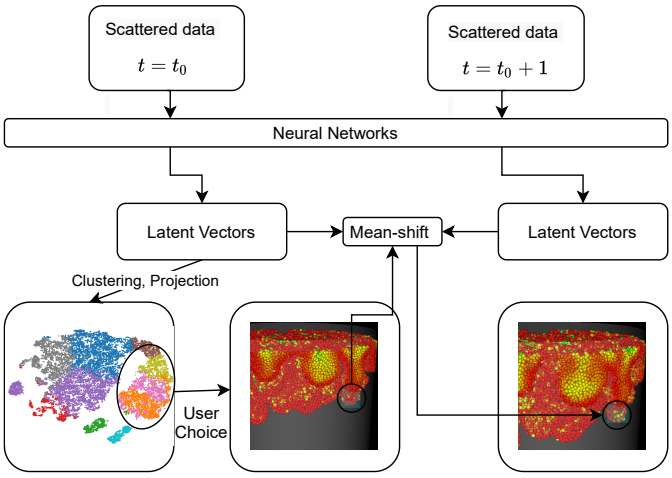

Fig. 1. Scheme showing our proposed approach. The neural network is pre-trained with particle patch samples. The tracking we show in the scheme is only for one time step. Actual feature tracking can be repeated until the desired time step is reached.

2.3 Mean Shift Tracking

The first application of mean-shift tracking was proposed by Comaniciu et al [9], where they track non-rigid objects in video clips by comparing color histograms between frames. It can achieve real-time performance with high accuracy, and so is widely used in computer vision applications. There have also been mean-shift applications for point clouds. Asvadi et al. [1] utilized mean-shift in 3D point cloud object tracking in the application of autonomous cars. In another work by Hermes et al [16], mean-shift tracking was used on 3D point cloud laser sensor data to track traffic participants. In our work, the mean-shift algorithm is applied in a high-dimensional latent space which is learnt by a neural network trained on scientific particle datasets.

3 Method Overview

In this work, latent vectors are produced to represent particle patches, and features are extracted and tracked based on the latent representations. A workflow of our approach is described in Fig. 1. Our approach can be described in the following steps.

First, a small portion of the data is chosen as the training samples, which are prepared into particle patches, i.e., groups of particles in local neighborhoods. With the training samples, an autoencoder is trained, which takes a particle patch as the input, produces a latent vector, and then reconstructs the input patch using the latent vector. Second, after the neural network is trained, we can use it to infer the latent vectors for particle patches at any selected time step in the domain. K-means clustering of the latent vectors is performed in the latent space to reveal plausible features. To help users visualize and interact with these latent features, t-SNE projection is performed to map the high-dimensional latent vectors to two dimensions, so that users can overcome 3D occlusion and easily circle the interested features. Finally, after the ~~interested feature clusters~~ are identified, the user can visualize them in the spatial domain and select a specific region that contains a feature of interest to track it over time. Features are tracked by maximizing the similarity in the distributions formed from latent vectors in local regions across time using the mean-shift algorithm repeatedly to find the matched location in the next time step until a desired time span is reached. Below we describe each step of our algorithm in detail.

4 Feature Autoencoder and Latent Vector

In this section we describe how we generate latent vectors for particle data sets using an autoencoder. As scientific particle data can be very large in size, to identify salient features in local regions, we first separate and process the particles into local patches. This not only reduces the memory working set size, but also makes it easier for the neural networks to learn the local representation of particles. A particle patch is defined as follows. Considering a 3D particle data with a d dimensional attribute, when a center location $\vec{p} \in \mathbb{R}^3$ is given, the corresponding particle patch $N(\vec{p},r)$ is defined by all the particles whose distance to $\vec{p}$ is less or equal to a distance r: $N(\vec{p},r) = \{\vec{q} \mid \|\vec{p}-\vec{q}\| \leq r\}$.

we can think of a particle as an instance of location-value pair of a function $f : \vec{p} \mapsto X_{\vec{p}}$, where $X_{\vec{p}} \in \mathbb{R}^d$ is the physical attribute at the particle location. The autoencoder is used to compress a particle patch into a smaller latent representation. Succinctly, the reason why the latent representations are smaller in size than the original particle patch are twofold. The first reason is that the autoencoder does not encode noises from the particle patch into the latent vector. The principle to determine noise is driven by the loss function of the autoencoder and the dataset. The second reason is that the encoder is capable of detecting redundant patterns among the training samples and succinctly encode them into a much smaller code as the latent vector. The second characteristic of the autoencoder also helps us extract essential features in the data. Patches with similar features will be in close proximity in the latent space, which makes it easier to group particle patches that exhibit similar features by clustering. Next, we provide the background about the autoencoder that encodes point clouds, and how it is modified for our specified application.

4.1 Particle Autoencoder

The original tasks for point cloud neural networks are to classify or segment point clouds, both of which need to extract features from the input point clouds. Unlike image data, point cloud data do not have mesh structures to connect the points, hence convolution kernels used for feature extraction like those in Convolutional Neural Networks (CNNs) cannot be easily applied. In this work, we adopt an elegant solution called GeoConv [21] whose core idea is to project the coordinates of a point into orthogonal directions and define a kernel on those directions. Here, we briefly explain the basic building block of GeoConv that is used to construct our autoencoder. In a particle patch $N(\vec{p},r) = \{\vec{q} \mid \|\vec{p}-\vec{q}\| \leq r\}$, vector $\vec{pq}$ is projected into three orthogonal bases $\{\vec{x},\vec{y},\vec{z}\}$, and the projected norm along each basis represents the "energy" in that direction. We aggregate the attribute of particle q into the center p based on its energy in different directions. To differentiate positive and negative directions, we define the following six bases in 3D: $B = \{(0,0,1),(0,0,-1),(0,1,0),(0,-1,0),(1,0,0),(-1,0,0)\}$. Then we define the aggregation of q's attributes to the center p as following:

$$g(\vec{p},\vec{q}) = \sum_{\vec{b}\in B} cos^2(\vec{pq},\vec{b}) W_{\vec{b}} X_{\vec{q}} \tag{1}$$

, where $\vec{b}$ is one of the bases vectors, $W_{\vec{b}}$ is the learnable weights in the corresponding direction and $X_{\vec{q}}$ is the attributes for particle q. A 2D example of this projection can be found in Fig. 2 (d). Finally we calculate a vector representation for this particle patch aggregating all particles in the patch according to their distance to the center. This process is demonstrated in Fig. 2 (b) weighted sum block.

$$GeoConv(\vec{p},r) = \frac{\sum_{\vec{q}\in N(\vec{p},r)} d(\vec{p},\vec{q},r) g(\vec{p},\vec{q})}{\sum_{\vec{q}\in N(\vec{p},r)} d(\vec{p},\vec{q},r)} \tag{2}$$

Here, distance function $d(\vec{p},\vec{q},r) = (r - \|\vec{p}-\vec{q}\|)^2$ gives more weights to the particles near the center. The full architecture of GeoConv is shown in Fig. 2 (b). We use the term shared fully connected layer to refer to a fully connected layer whose weights are shared among different particles. This means the weight matrix in GeoConv architecture shared FC is 64×256 in size, and each of the n particles are multiplied with the same matrix. Shared FC is efficient and can be implemented with 1×1 convolution.

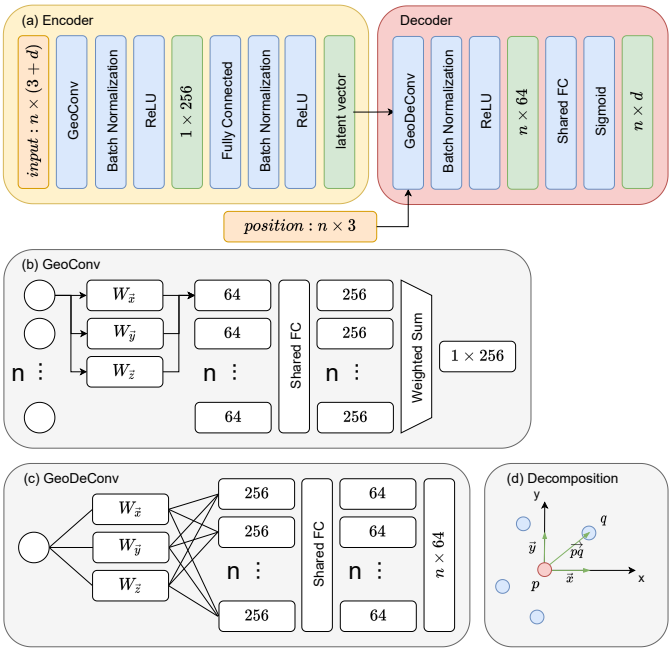

Fig. 2. (a) The architecture of our autoencoder. The input is given in a particle patch with n particles, each of which has 3 dimensional position and d dimensional attributes. (b) GeoConv and (c) GeoDeconv architecture is given in additional blocks. Shared FC stands for shared fully connected layer. (d) Decomposition blocks shows a 2D example of GeoConv, where $\vec{pq}$ is projected onto two orthogonal bases.

Unlike point cloud classification or segmentation, our particle data do not have any labels to indicate whether the particle patch is the desired feature. Therefore, we use an autoencoder scheme. We present the architecture of our autoencoder in Fig. 2 (a). A classical autoencoder is composed of two parts, the encoder and the decoder. The encoder takes a particle patch $N(\vec{p},r)$ and the corresponding attributes $X_{\vec{q}}, \vec{q} \in N(\vec{p},r)$ as input and produce a single latent vector $l_{\vec{p}}$.

$$enc(N(\vec{p},r), \{X_{\vec{q}} \mid \vec{q} \in N(\vec{p},r)\}) \mapsto l_{\vec{p}} \tag{3}$$

Since the particle patch size is relatively small in our experiments, we only use one layer of GeoConv followed by one fully connective layer.

A classical decoder will simply take $l_{\vec{p}}$ as input and reconstruct the particle patch back. One observation is that, for particle data, features are usually defined by the attributes of the particles, and hence reconstructing the locations of all particles in the patch is not essential for the decoder whose primary task is feature extraction. Therefore, as a modification to the point cloud autoencoder, we decouple the particles' position and attribute values in our autoencoder. Our modified decoder takes the latent vector and particle positions from the patch as an input and only predict the values at those positions.

$$dec(l_{\vec{p}}, N(\vec{p},r)) \mapsto \{X_{\vec{q}} \mid \vec{q} \in N(\vec{p},r)\} \tag{4}$$

With this modification, we can enter any particle positions in the patch for the decoder to predict their physical attributes. This new decoder architecture can be constructed using the idea of GeoConv. We treat the latent vector as the attribute of the center $\vec{p}$. The new attribute on neighboring particles can be calculated by swapping $\vec{p}$ and $\vec{q}$ in Equation 1. We call this new layer in the decoder GeoDeconv, whose detail is also shown in Fig. 2 (c). Our decoder is constructed using one layer of GeoDeconv followed by one shared fully connective layer. Given a latent vector $l_{\vec{p}}$, since now we can provide any particle positions to the decoder and make it predict the attributes at the provided position, it becomes easy to volume render the features represented by the latent vector by simply providing the regular grid positions and having the attributes predicted at the grid points.

Some other details of our architecture: Batch normalization is applied after each layer to stabilize the training and make the convergence

faster [18]. We use ReLU as the activation function except for the last layer [29]. Sigmoid is used at the last layer to constrain the output value between $[0,1]$. Mean squared error between the ground truth attributes and predicted attributes is the loss function we use.

4.2 Training Data Sample

In our experiments, we took a small portion of all particles as the training data. Instead of simply performing random sampling, a special sampling rule is used considering the distributions of the particle attributes by drawing less samples with values that occur very frequently, and more samples that have less frequent attribute values. This sampling rule is based on the observation that very frequent attribute values usually correspond to the background or homogeneous regions with nearly no features. In addition, it is of no benefit to train the neural networks with redundant training data. This strategy is also suggested by a recent work of probabilistic data-driven sampling [4]. To realize this sampling strategy, we used a similar method as in [4] in the particle attribute space to perform the sampling. As for neighborhood definition, Choosing the k-nearest neighbors and choosing points in a radius r are both common ways to define neighborhoods in point cloud neural networks. In recent literature [17], defining the neighborhood with a radius r is proven to be superior because it maintains more consistent feature size. Therefore this approach is used in our work. To achieve fast neighbor search, we perform a spatial partitioning of the data set in every time step using a kd-tree [3]. Since we are only interested in the particles' local relative positions to the center but not their absolute positions, the coordinates of particles are first normalized to be in the range $[-1,1]$, and then particles in $N(\vec{p},r)$ are translated relative to the center particle $\vec{p}$. In our experiment, we choose training batches to be as large as our memory constraint allows given the dimensionality of our data attributes. We discuss the choice of r and latent vector dimension v later in the experiments section.

After the neural networks are trained, it can be used to generate latent vectors for any given particle patch. The resulting latent vectors will be used for feature extraction and tracking, which is discussed in the next section.

5 Feature Extraction and Tracking

We have described how to succinctly represent the particle patches with latent vectors. In this section we introduce our feature extraction and tracking methods in detail.

5.1 Latent Space Exploration

The latent vector produced by the neural network is a compact feature descriptor for a particle patch. Therefore patches with similar features will be in close proximity in the latent space. This enables us to explore and find features by analyzing the clusters in this space. In our approach, cluster analysis is done by k-means clustering in the high-dimensional latent space, and a t-SNE projection of latent vectors to 2D is used to help user visualize and interact with the distribution of latent vectors. We choose k-means as our clustering algorithm mainly because of its efficiency on large data.

The first step of exploration is to select a time step of interest by the user. This can be done based on the prior knowledge about when the interesting features may occur. Or the user can simply select once every few time steps to explore. To reduce the computation cost for clustering and projection of large amounts of latent vectors, instead of computing the latent vectors for patches around every particle in one time step, we choose particle patch centers with a regular interval, which can be covered by the r sphere similar to what is used for network training. This greatly reduces the number of latent vectors and will not impact the clustering result much, since adjacent particle patches tend to have similar latent vectors. We cluster the latent vectors in their original space using a standard k-means algorithm and color the projected point with the k-means cluster id. Plausible features are already segmented and identified by clustering. However it is difficult for users to explore the clusters to find the clusters of interest in the high dimensional latent space or in the 3D spatial domain, especially when the number of clusters is large.

To alleviate this problem, latent vectors are first projected to 2D space using a standard implementation of t-SNE projection, and then users can visually find clusters and circle the interested ones as shown inFig. 1. This projection also helps the user determine the number of clusters used in k-means, which will have impact on the feature extraction result. For example, if the number of clusters is set to too high for k-means, many small clusters will be produced, and in the 2D space no clear boundaries can be seen to differentiate different features. Users can increase or decrease the number of clusters based on this projection view, so that a more accurate clustering can be achieved.

Patch-wise feature extraction gives us a coarse segmentation of the features. To obtain a particle-wise feature segmentation, an average latent vector is calculated for the circled clusters in the last step. We then infer the latent vectors for the patches around every particles and filter them using its distance to the average vector. Since our network size is relative small, it is shown in the result that inference of millions of points can be finished in reasonable time.

5.2 Mean-shift Tracking Using Latent Distribution

The aforementioned feature extraction method involves identifying features of interest from clusters of latent vectors at the time step of interest. Once the interested region is found, the next task is to know how the feature develops in the future time steps. Traditionally, people perform feature tracking by first having feature extracted in every time step and then matching them across the time steps. In that case, the tracking efficiency is bounded by the extraction efficiency and can be expensive. The situation becomes worse when the latent inference is time consuming. Therefore we propose a feature tracking approach based on tracking the latent similarities on the fly between time steps.

First of all, a region of interest is selected by the user as mentioned in the previous section, where a bounding box in the spatial domain can be computed. This can be easily done since features are already identified in the current time step of interest. Users can simply find and select one or several features they want to track over time. The v dimensional latent vector for every particle in the cubic area is collected. We first reduce the latent dimensions to four using PCA so that only the most important dimensions of the latent vectors in PCA space are used to reduce the computational overhead of tracking. Treating each latent vector as a sample, our feature of interest selected by the user can be represented by a four-dimensional histogram. Therefore our tracking goal between two consecutive time steps is to identify the cubic area that has the most similar latent histogram in the later time steps. Since we do not know the full range values in the latent vector, we determine the ranges for histogram building dynamically based on samples in the current and the previous time steps. Initially we assume the spatial location of the feature is the same as that of the last time step, which is used as the starting point for tracking. Then we move the cubic region to the most similar position using the mean-shift algorithm introduced by Comaniciu et al [9]. Repeating the search step on the time sequences will give us the movement of the interested feature through time steps. The user can also choose multiple regions to track. Tracking multiple features can be achieved efficiently through parallel computing.

The high level idea of mean-shift tracking is that first we give different weights to the particles in the interested region according to the comparison between their latent vector with target latent distribution. And then the area is shifted to the higher weighted direction. This is repeated until the area does not move. We follow the same technique in [9], where they use Epanechnikov kernel to smooth the mean-shift optimization space. There are two hyperparameters we need to determine for tracking. The first is the number of bins to build the latent histogram. Since our latent space is high dimensional and samples are relatively sparse, we set it to a small number. The second is h in the Epanechnikov kernel, which is a smoothing parameter to prevent the tracking process to get stuck in local optima. The influence of these hyperparameters will be discussed in the experimental results.

6 Experiments

This section demonstrates how our proposed extraction and tracking algorithm works on two particles datasets. The first dataset is from

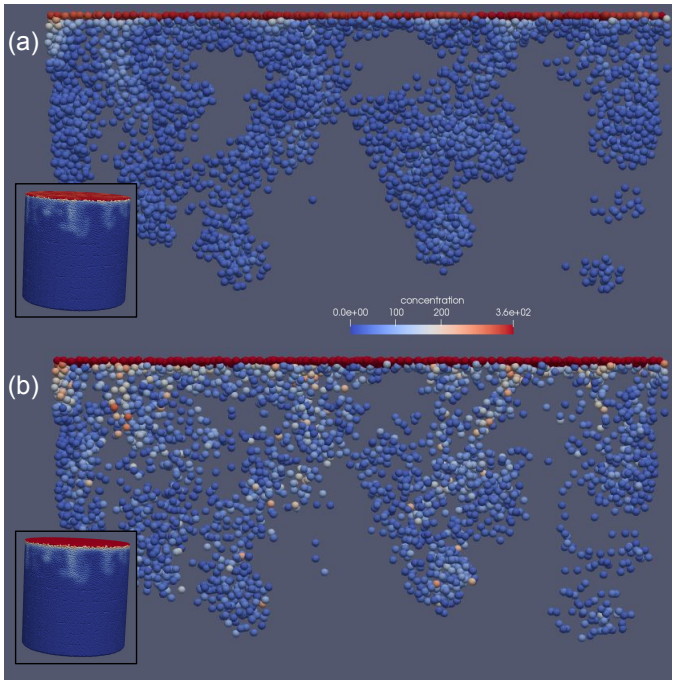

Fig. 3. (a) Reconstructed (b) Original. Smaller images in the corner show the full data in one time step. The larger images show the particles filtered with the same concentration threshold. The color of the particle denotes its concentration value.

2016 scientific visualization contest [12] and the second is from 2015 SciVis contest [8].

Our experiments are performed on a machine with an 8-core Ryzen 2700X CPU, 32GB RAM, and an Nvidia RTX 2060 GPU. The autoencoder is implemented using the Pytorch framework, and the mean-shift algorithm is implemented in Python.

6.1 Case Study 1: Salt Dissolution

The first case study dataset is from a salt dissolution simulation using the Finite Pointset Method. The basic simulation setup consists of a cylindrical flow domain that contains water [12]. At the top of the cylinder, modeled by a corresponding boundary condition, is a solid body of salt dissolved by the water. Each simulation of the provided ensemble is based on this setup. Due to the transient nature of the solution process, approximately 120 time steps are provided for each simulation to provide significant temporal resolution. The original data has 3 different resolutions and the resolution we used has around 190,000 particles at every time step. Every particle has a 4 dimensional physical attributes: a concentration value and 3 dimensional velocity. These particles are distributed in a cylindrical space whose radius is 5 units and height is 10 units. The feature extraction and tracking goal is to find the finger-like high salt concentration areas.

6.1.1 Autoencoder Training and Reconstruction

To capture the feature variance in all 120 time steps, using the method discussed before, we sampled around 1500 particles in each time step. We treated the coverage of all samples in 120 time steps in training as one epoch. We ended the training at epoch 15 when the loss no longer dropped. To verify the quality of our trained autoencoder, we performed a comparison between the reconstructed and the original particles. We chose the particle patch centers every 0.33 units in all three spatial dimensions for reconstruction, while the particle patch radius was set as 0.5. The distance between the particle patch centers was chosen to be smaller than the radius to remove the seam artifacts. Overlapping particles' reconstructed value was averaged between the patches. In Fig. 3, we compare a zoomed-in slice along the z-axis. This slice was filtered to show only the particles with concentration values higher than 25. We can observe that the reconstructed concentration values are smoother among the reconstructed particles. It was found in the study [12] on this dataset that these high frequency concentration fluctuations are noises and are to be removed before feature extraction. Therefore this smoothing effect from the reconstruction is beneficial to our subsequent feature extraction. The peak signal-to-noise ratio (PSNR) between the reconstruction and original data is 26.9. Considering that we only rely on the clusters of latent vectors to extract feature but not to replace the original data with the reconstructed one, PSNR values in this range is acceptable to our goal.

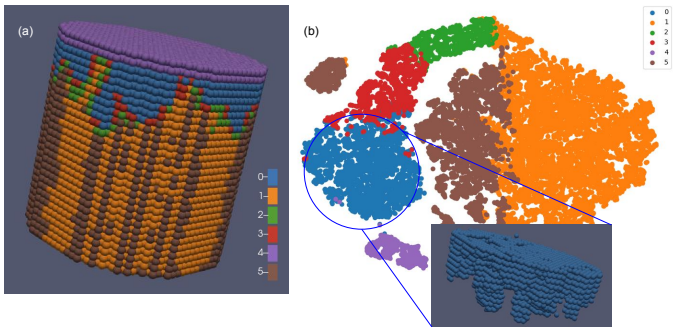

Fig. 4. (a) we represent coarse particle patches with a sphere and color it by the cluster id. (b), we show the t-SNE projection of the latent vectors also color them with cluster id. Cluster id is obtained using k-means in the high-dimensional latent space. (a) and (b) use the same color scheme for clusters. It can be seen that blue cluster is the viscous finger cores. Red and green clusters are at the boundary of viscous fingers. Yellow and brown clusters are low concentration background. Purple cluster is the high concentration area where the salt is injected.

6.1.2 Feature Extraction Results

We first choose time step 25 as our feature extraction example and the starting point of tracking, where according to the prior studies [12] on this dataset, the number of viscous fingers start to increase and their structure start to become complex. Particles patches are chosen using the same method presented earlier and a latent vector is generated for each patch. After we got the coarse latent representations, k-means algorithm was applied to assign a cluster id to each patch. In Fig. 4 (a), we show the k-means clustering result by coloring each block particle patch with their cluster id. It can be observed that features of viscous fingers and low concentration background is separated into different clusters. However due to 3D occlusion and the existence of multiple clusters, it is difficult to examine the extraction result directly in 3D visualization. We use t-SNE projection as an assisting tool to help this cluster analysis. Latent vectors are projected to 2D using t-SNE and colored by their cluster id as shown in Fig. 4 (b). This projection provide us with a clearer view of these clusters. We can see that even though brown and yellow clusters are separated in k-means, they are close to each other in t-SNE projection and, indeed, they are all background blocks in the dataset. This also happens for green and red clusters, where they are both boundary patches around the fingers. These effects indicate that we may have chosen a cluster number that is too high for the k-means algorithm. Users can circle in the projected space and visualize in 3D to further verify the cluster's feature. In this example, the blue cluster is exactly corresponding to the viscous fingers in the dataset. After identifying the clusters of interest, we inferred the latent vector for a particle patch centered at every particle and calculated its cosine distance to the interested cluster center to get a high resolution extraction result. Particle patches whose latent distance to the cluster of interest is lower than a threshold is chosen to be the extracted features. We show the extracted fingers in Fig. 5 (b) and compare our extraction with other approaches in the later sections.

In training the autoencoder to generate latent vectors, we need to choose two hyperparameters, the size of neighborhood r and latent vector dimension v. The different choices are tested through the experiments. Firstly, we tested the influence of r on the extraction results. In Fig. 5 (a), it shows the particles of 4 extraction results from the same time step with $r = 0.1, 0.3, 0.5, 0.8$ respectively. Small r values will generally introduce more noises to the extraction result, which can

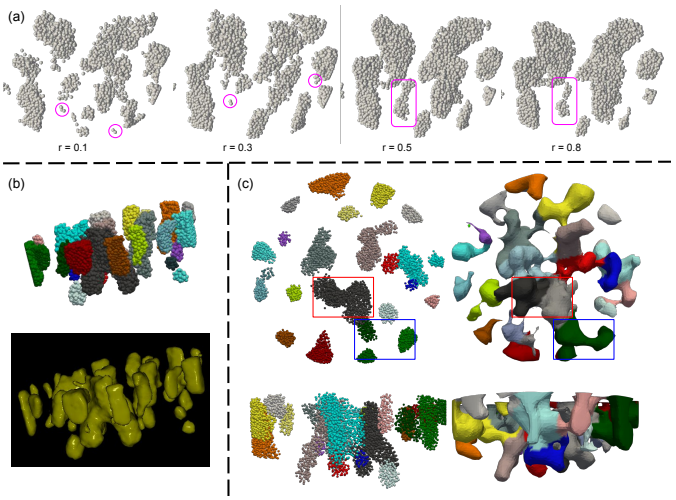

Fig. 5. (a) Extraction results using four different models with different size of neighborhood considered. Notices noises (purple circles) when $r = 0.1, 0.3$, and finger shrinking effect when $r = 0.8$ (purple squares). (b) Extracted fingers shown as particles and surfaces. (c) Different views for comparison between finger extraction through latent vectors and topology analysis. Fingers are colored to best match the same fingers from 2 methods. It is observed 2 methods have different separation of fingers. Some fingers are grouped into one in our methods but divided using topology analysis (red square). Some fingers are in the reversed situation(blue square).

be seen as fragmented fingers when $r = 0.1$ and $r = 0.3$. Meanwhile, larger r will make small fingers shrink or even disappear, where an example can be seen in the purple squares. We attribute this effect to that larger neighborhood will include the finger boundary in the feature particle patches, thus small fingers will be smaller or eliminated. Based on these observations, we chose $r = 0.5$ in our following experiments.

In the next step, we test how v , the dimensionality of the latent vectors, should be chosen. Generally speaking, v is the bottleneck of the autoencoder, thus controls the network's learning capacity. Hereby, we introduce the intrinsic dimension, which describes how many dimensions are needed to generate a good approximation of the data set. In order for the latent vector to be a good particle representation, v should be set larger than the intrinsic dimension of the data set. We plot all latent vectors in one time step using the parallel coordinate plots at $v = 16$ and $v = 32$ in Fig. 6. It can be observed that when $v = 16$, there are 3 dimensions that have little variance, and when $v = 32$, there are 16 dimensions that have little variance, which makes these dimensions redundant in describing the input. Therefore we hypothesize that the intrinsic dimension of this data set is less than 16. Setting $r = 0.5, v = 16$, our feature extraction results are shown in Fig. 5 (b), as direct particle rendering and surface extraction. In the direct particle rendering figure (top), since latent vector clustering only extracts finger cores from the background but not separates different fingers, different finger cores are separated spatially with DBSCAN clustering algorithm and different colors are assigned to them. Surface extraction is done by processing the particle patches defined at a regular grid, and assign 1 to patches of finger cores, -1 to patches of other clusters and extract iso-surface at the value 0 using marching cubes.

6.1.3 Compare with Other Ways to Represent Particle Patches

Using autoencoder to produce feature descriptors is not the only way to represent a particle patch. In this section we compare our latent representation with two other methods: representing the patch with neighborhood mean and with principle components. One prior knowledge about the viscous fingers is that they only appear in the area that has relatively high salt concentration. A straightforward way to extract fingers is to filter the particles with a concentration threshold. Therefore, the first method we compare is averaging the concentration in the neighborhood to remove the high frequency noises and extract features with a threshold. The comparison result is shown in Fig. 7. It

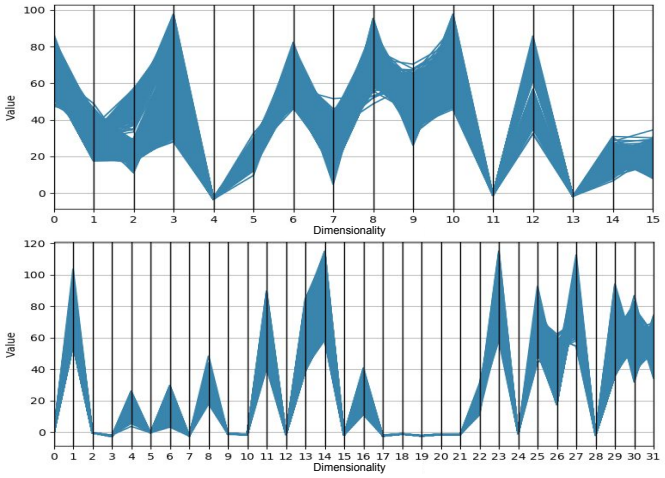

Fig. 6. All latent vectors from one time step is plotted with parallel coordinates. The top figure is from model with latent dimension $v = 16$. The bottom figure is $v = 32$. Intrinsic dimension is less than the number of latent dimensions with variances.

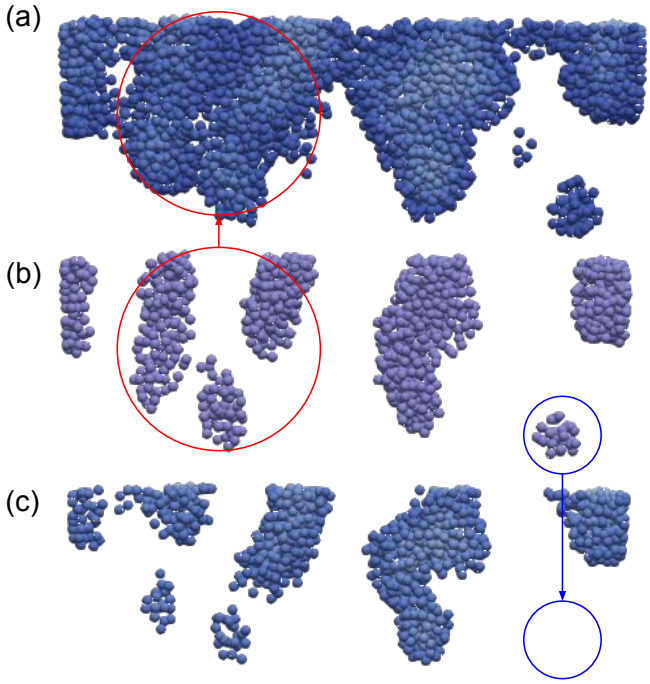

Fig. 7. (a) Average the particle's concentration with particles in local neighborhood and set a low concentration threshold to extract feature. (b) Feature extraction result with latent vectors. (c) The same average set as (a), however set a high concentration threshold. Red circle area shows low threshold will merge different fingers so that we can not distinguish them, blue circle shows high threshold will omit some features compared to the results generated by latent vectors.

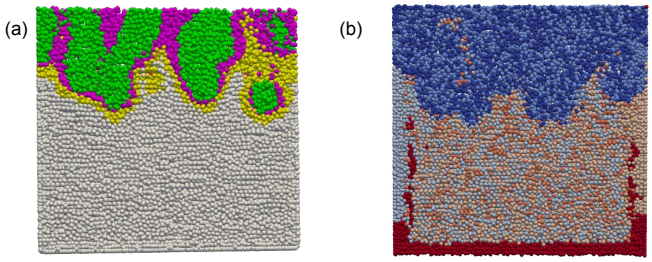

Fig. 8. Feature extraction with k-means clustering using (a) latent vectors and (b) principle components of particle patch. Latent vector cluster clearly show the finger cores. However clusters of principle components only vaguely show the boundary of low and high concentration areas.

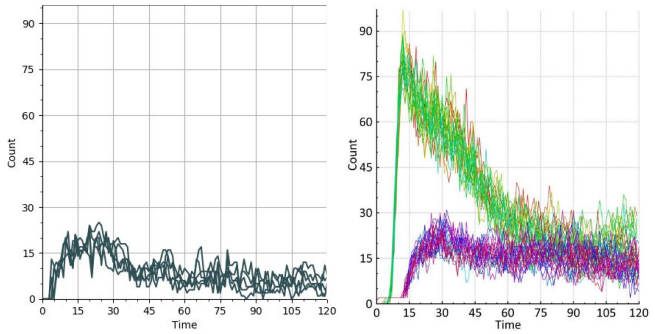

Fig. 9. Left: The extracted finger count across all 120 times on 5 different ensemble runs using our method. Right: Finger count from 2016 scivis contest winner [12]. Blue and purple lines are from the heuristic clustering method in their work, while green and yellow lines are from topology analysis.

demonstrates the difficulty to determine the threshold if we only remove the noises with averaged concentration in the particle patches. Low thresholds will make it harder to distinguish different fingers, while high thresholds will make some fingers disappear. Our neural network method has the advantage of capturing features defined by the gradients automatically. It is probably true that we can identify features by calculating concentration gradients and there has been previous work [43] using gradients to identify viscous fingers. However in [43] the authors use regular grid data and for particle data, calculating gradients is difficult and can be expensive to apply mesh-free methods to achieve this on particles.

Another way is to perform principle component analysis (PCA) on the particle patches and represent each particle patch with their principle components. Principle components can be seen as a linear regression of the particles in the patch. In Fig. 8, we can see the cluster of principle components only vaguely show the boundary between high and low concentration areas. This is due to (1) Principle components are influenced by the positions of particles in the patch, so that patches with different particle distributing patterns will fall into different clusters.(2) PCA is only a linear transformation of the neighborhood and does not consider any global information between the particle patches.

6.1.4 Compare with Topology Analysis

The approach that we compare in this section is from the winners of 2016 SciVis contest, who proposed two approaches to extract finger structures [12]. The first one is based on topology analysis on the data and the second one uses slice-wise clustering. We ran the topology based method and our method on the same data from time step 25 of a single simulation from the ensemble. The finger extraction results is shown in Fig. 5 (c). Roughly speaking, the two methods generate similar numbers of fingers, 24 for the topology analysis and 25 for our method. We colored the fingers to match the same fingers in the two methods. However, there are still some noticeable difference between the two results.

One observation from the result is that topology analysis generates fingers of larger size. It is mostly because the analysis is based on a smooth distance field, which will connect the boundaries between fingers. Secondly, topology analysis and our method separate the fingers differently in space. Since we use DBSCAN in our method to separate them, connected fingers having the same particle density will be identified as one finger. This effect can be seen in Fig. 5 (c) within the red squares. On the other hand, the topology analysis only uses persistence thresholds to find fingers, which will fail to detect some and merge the corresponding region to others as what in the blue squares show.

Quantitatively, we compare the number of fingers extracted by our method with the method in [12] across all time steps in Fig. 9. It can be seen that our method have similar finger count trend with their two methods. Finger numbers both go up at first 30 time steps and then drop when the data become chaotic in the latter time steps. The finger count in this paper is similar to the heuristic clustering result in theirs. It is explained in their work that the number difference is mainly because of the choice of threshold. Generally speaking, our method have comparable feature extraction results both qualitatively and quantitatively with the winner method for this SciVis contest dataset.

6.1.5 Tracking Results

As mentioned in the method section, we need to choose two parameters for feature tracking. The first one is the radius of Epanechnikov kernel and the second is the number of bins b in each dimension for histogram building. We did experiments on different radius and find that the choice of kernel size should be based on the actual feature size. As for b, we found any number larger than 2 in each dimension will cause instability in the tracking. Setting $b = 2$ will give us 16 histogram bins in total, since we have four latent dimensions after PCA projection. The experiment results can be found in the supplementary materials. The termination criteria for the mean-shift algorithm is when the selection window's shift distance is less than 0.0001, or after the algorithm reaches 100 iterations. In Fig. 11 we show the number of iterations before the algorithm ends for the 4 tracked cases. All tracking examples terminate before reaching the iteration limit we set showing that the algorithm converges well.

We select 2 fingers from time step 25 in the same run from the ensemble simulation to demonstrate our tracking method. The tracking result is shown in Fig. 10. We show the tracking result every 5 time steps and the animation can be found in the supplementary materials. The first tracking does not go off the feature through 20 time steps. The second tracking sticks to the feature before the feature starts to deform and finally disappear beginning at the time step 35. Generally speaking tracking small features in the dataset is more challenging because our method relies on the overlap between time steps.

6.1.6 Time and Memory Consumption

Under the hyperparameters chosen as discussed, our training of autoencoder described above took 10.5 minutes per epoch, which made the total training time of 15 epochs to be 2.7 hours. Once the autoencoder is trained for this data set, inference of latent vectors takes 75 seconds per million particles on the same machine. Since one time step consists of about 190,000 particles. It takes approximately 15 seconds to inference all particles in one time step and produce high-resolution extraction result in the experiment. Considering I/O, data-preparation and clustering, the total feature extraction time for one time step in this data set is around 20 seconds.

As shown in the Fig. 11, the feature tracking time depends on the size of feature of interest. Larger feature size means more latent vectors to be generated. The total tracking time is dominated by the latent vector generation time. In the case that the feature of interest is a finger tip, which is consisted of approximately 200-400 particles, the tracking time is around 90ms. This is much more efficient than the method where features are first extracted and then matched in 2 time steps.

As for the memory consumption, our method processes the whole dataset in a patch-by-patch manner. The memory need for latent generation can be almost neglected. The most memory consuming part of our approach is the t-SNE projection and clustering. However we can increase the distance between particle patch centers to reduce the the number of patches in this step to alleviate this problem. Once the interesting cluster is found we can always produce high-resolution extraction result afterwards.

6.2 Case Study 2: Dark Sky Simulation

Our second case study uses a cosmology simulation data set [8]. This particle data set is composed of 100 time steps with around 2,000,000 three-dimensional particles in each time step. The scale factor is different for every time step. However, after normalizing the particles' positions, they are distributed in a $62.5 \times 62.5 \times 62.5$ Mpc/h (MegaParsecs divided by dimensionless Hubble constant) space. This unit is also the distance unit used in the provided halo list [8]. Every particle has its position vector, velocity vector, acceleration vector, and gravitational potential energy *phi*, making it a 7 dimensional physical attribute.

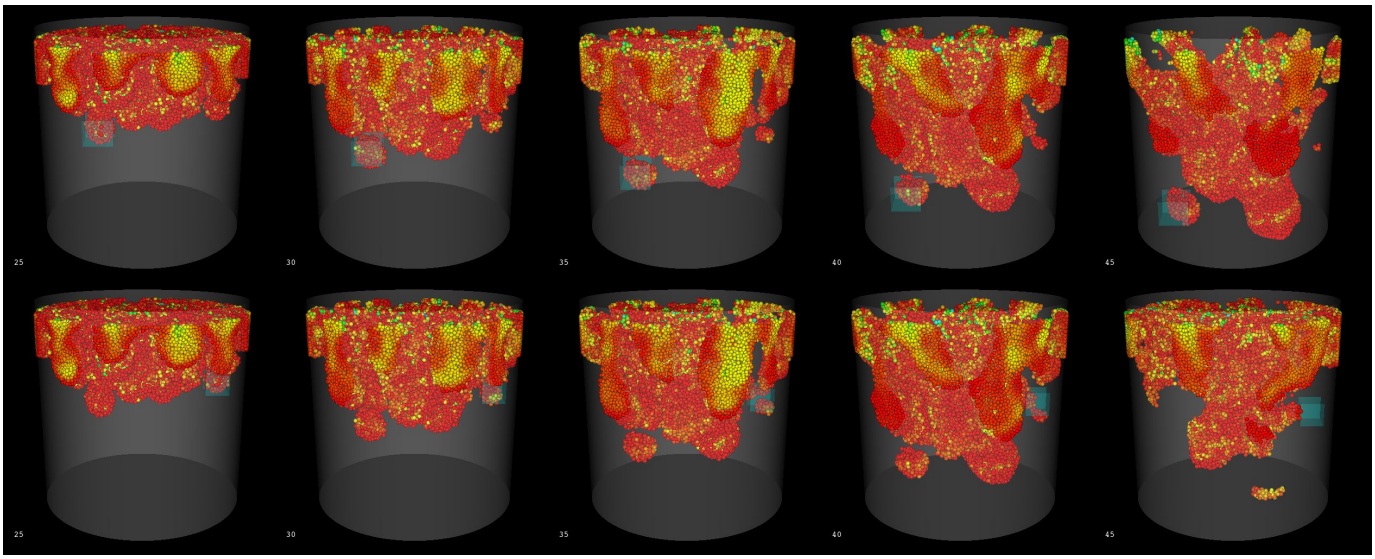

Fig. 10. Two different tracked features every 5 time step through 20 time steps in two rows. Time is labeled at the left bottom corner of the image. The user selection of the interested feature is highlighted by the transparent blue square in the figure. Particles are colored by their concentration values.

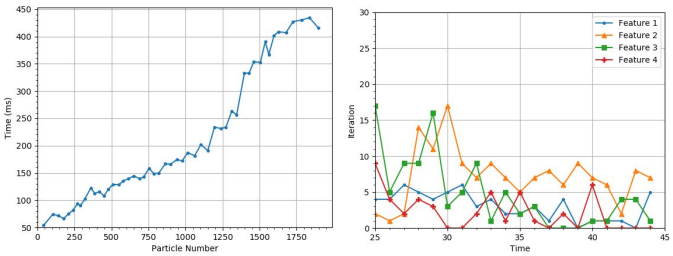

Fig. 11. Left: Relations between the number of particles in the feature area and feature tracking time . Right: Number of iterations before convergence for 4 different feature tracking examples. Feature 1 and 2 are the same features as shown in Fig. 10.

Table 1. The loss values are shown for models trained with different parameters. "-" indicates that the parameter combination was not tested.

loss	$v = 256$	$v = 512$	$v = 768$
$r = 0.2$	0.15	-	-
$r = 0.4$	0.10	0.13	-
$r = 0.6$	0.18	0.12	0.13
$r = 0.8$	-	0.37	-

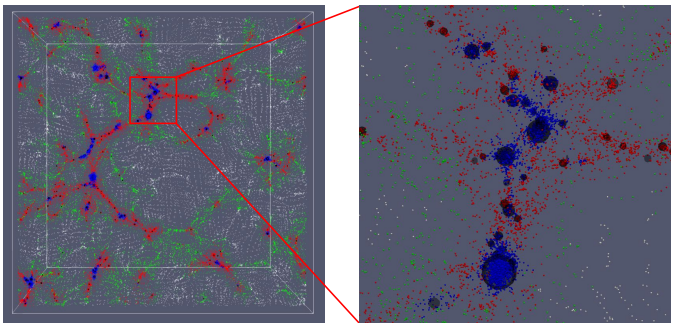

Fig. 12. One slice of the data from time step 49. Particles falling into four feature clusters are shown in four different colors. We show the actual halo position and size with transparent black spheres. It can be seen most of halos are located in the blue and red clusters area.

Along with the raw particle information, the data set comes with a list of halos, which is a complex feature defined by the particle density and other physical attributes. The halos are found by the method proposed by Behroozi et al [2]. Besides the position, size and other derived physical attributes of the halo, the halo list also provides the correspondence of halos between time steps, i.e., how each halo moves in consecutive time steps. We use the halo list to verify our feature extraction and tracking result, as each halo is labeled and tracked over time. The purpose of this study is to show that our approach can perform feature extraction without any prior knowledge.

6.2.1 Experiment Setup

We followed the similar set up to train our autoencoder, taking a subset of all the particles as the training data. The input to the autoencoder are the 3 positional dimensions and 7 physical attribute dimensions. Following the sampling strategy discussed in Sect. 4.2, we sampled 10,000 particles in every time step, which sums up to 1,000,000 samples in total. The training ended after 35 epochs when the loss value converged.

We determined the latent vector length v and the neighborhood size r for the autoencoder by observing the loss values in the test phase with different hyperparameters. The testing data set was generated by sampling 1% of particles from every time step, which was approximately 20,000 particles in each. The loss values were averaged across time steps. Testing results with different setups are shown in Table 1. Some combinations were not tested, for the trend in those tested cases already indicated poor performance. According to the results, when r is 0.4 or 0.6, it gives significantly lower testing loss than r being 0.2 or 0.8. We chose $r = 0.4$ other than $r = 0.6$ for inference efficiency. As for the latent vector length, we did not observe any accuracy gain when v is increased from 256 to 512, therefore we set $v = 256$.

6.2.2 Extraction Results

After the autoencoder is trained, we went through the clustering and t-SNE projection to find the interesting features. For the simplicity of presentation, we directly show the latent vectors for patches centered at every particles of time step 49 and their clustering results. We use the k-means algorithm and choose the number of clusters to be 4. Fig. 12 shows each particle colored with their cluster id. To compare with the ground truth features, we draw the real halos as transparent black spheres on the right of the figure. On the left image, blue and red cluster particles demonstrate the interesting areas of high density. The right image shows that halos mostly appear in these high density areas. Quantitatively, we calculate the percentage of halo particles that are covered by these two clusters. The blue cluster in the figure covers 70.3% of ground truth halos, while combining the red and blue cluster covers 98.8% of all halos.

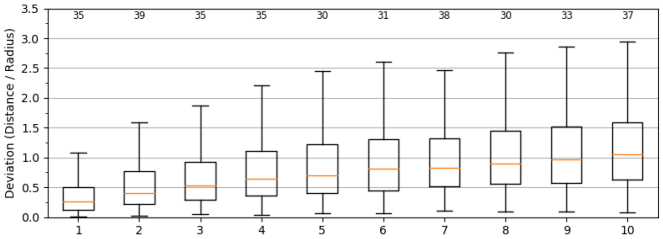

Fig. 13. Tracking deviation for 10 time steps from 594 random halos shown by boxplot. The number over the upper whisker is the count of outliers whose deviation is larger than the upper whisker for the timestep.

6.2.3 Tracking Results

Since we had the features' ground truth positions, it is easy to evaluate our feature tracking results by calculating the distance from the center of the extracted feature produced by our algorithm and the real feature center. Taking the feature size into account, we use $\frac{d}{r}$ as the feature deviation metric, where d is the distance between the centers and r is the ground truth halo radius. If this metric is larger than 1 but smaller than 2, the center of our tracked halo is no longer within the ground truth halo, meaning a slight deviation. If larger than 2, the tracking center and the feature center will have no overlap. We randomly chose 594 halos from the halo list and tracked them for 10 time steps. The results of the tracking error are shown in Fig. 13. The median deviation generally increases with time and reaches one after ten time steps. After tracking for 10 time steps, a majority (approximately 75%) of the halos have a deviation below 1.5, indicating reasonable performance over time. The upper whisker in the boxplot is set at $Q3 + 1.5 \times (Q3 - Q1)$, where $Q3$ and $Q1$ is the third and first quartile respectively. The number of outliers whose deviation is larger than the upper whisker is about 35 (6% of tested cases).

6.2.4 Timing

The inference time for our model using the optimal parameter settings is 81s per million particles. The efficiency of our inference can be further increased with more powerful GPUs. Since there are around two million particles in each time step of this data set, feature extraction for each timestep takes approximately 170s which includes I/O and data preparation. The tracking time follows a similar trend as the performance shown in case study 1. The average time used for one-time-step tracking in our tested features is 107 ms.

7 Conclusion and Future Work

We present a particle feature extraction and tracking algorithm based on the latent vectors generated by an autoencoder. Our method processes particle data directly avoiding grid re-sampling which can be lossy and expensive. We demonstrate successful feature extraction results in two different particle datasets. In the first dataset, we show our latent vector can reconstruct particle patches with less noise from the original input. The finger structure extraction result is superior to the concentration threshold method or representing the block with principle components. We compare our extraction with the SciVis Contest winner approach and comparable results are shown. Our tracking approach was verified by manually picking finger structure tips and follow the tracking result temporally, where the method worked well as long as feature regions have overlap between time steps. In the second dataset, our approach is adapted to a larger particle data with multi-dimensional physical attributes. Still, interesting features are captured by the latent representation. Even though the particle segmentation is not completely the same as the original halo finding algorithm, our method provides an efficient way to extract related regions for tracking and further analysis. Tracking is quantitatively analysis in this dataset using the ground truth halo positions. Our tracking results are stable for most halos in this dataset.

One drawback of our approach is our feature tracking algorithm only considers the local similarity across time steps for efficiency reason. This may potentially lead to the problem that our tracking result is not globally optimal and we cannot identify feature changing events like split or merge along time steps. However if the feature tracking efficiency is not the priority, one can use other proposed method like [26] to solve these problems. Use neural networks to solve this temporal global optimization problem is another possible way. This includes modifying the neural network to recurrently take inputs from multiple time steps and generate latent vectors for spatial-temporal features, which is a future work of our study.

References

[1] A. Asvadi, P. Girão, P. Peixoto, and U. Nunes. 3d object tracking using rgb and lidar data. In *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*, pp. 1255–1260. IEEE, 2016.

[2] P. S. Behroozi, R. H. Wechsler, and H.-Y. Wu. The rockstar phase-space temporal halo finder and the velocity offsets of cluster cores. *The Astrophysical Journal*, 762(2):109, 2012.

[3] J. L. Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.

[4] A. Biswas, S. Dutta, E. Lawrence, J. Patchett, J. C. Calhoun, and J. Ahrens. Probabilistic Data-Driven Sampling via Multi-Criteria Importance Analysis. *IEEE Transactions on Visualization and Computer Graphics*, 14(8):1–1, 2020. doi: 10.1109/tvcg.2020.3006426

[5] J. Chen, D. Silver, and L. Jiang. The feature tree: Visualizing feature tracking in distributed AMR datasets. *PVG 2003, Proceedings - IEEE Symposium on Parallel and Large-Data Visualization and Graphics 2003*, pp. 103–110, 2003. doi: 10.1109/PVGS.2003.1249048

[6] Z. Chen, W. Zeng, Z. Yang, L. Yu, C. W. Fu, and H. Qu. LassoNet: Deep Lasso-Selection of 3D Point Clouds. *IEEE Transactions on Visualization and Computer Graphics*, 26(1):195–204, 2020. doi: 10.1109/TVCG.2019.2934332

[7] 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. doi: 10.1109/TVCG.2018.2796085

[8] T. Christoudias, C. Kallidonis, L. Koutsantonis, C. Lemesios, L. Markou, and C. Sophocleous. Visualising the dark sky IEEE SciVis contest 2015. *2015 IEEE Scientific Visualization Conference, SciVis 2015 - Proceedings*, pp. 79–86, 2016. doi: 10.1109/SciVis.2015.7429496

[9] D. Comaniciu, V. Ramesh, and P. Meer. Real-time tracking of non-rigid objects using mean shift. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2(7):142–149, 2000. doi: 10.1109/CVPR.2000.854761

[10] S. Dutta and H. W. Shen. Distribution Driven Extraction and Tracking of Features for Time-varying Data Analysis. *IEEE Transactions on Visualization and Computer Graphics*, 22(1):837–846, 2016. doi: 10.1109/TVCG.2015.2467436

[11] G. Favelier, C. Gueunet, and J. Tierny. Visualizing ensembles of viscous fingers. In *IEEE Scientific Visualization Contest*, 2016.

[12] P. Gralka, S. Grottel, J. Staib, K. Schatz, G. Karch, M. Hirschler, M. Krone, G. Reina, S. Gumhold, and T. Ertl. 2016 IEEE Scientific Visualization Contest Winner: Visual and Structural Analysis of Point-based Simulation Ensembles. *IEEE Computer Graphics and Applications*, 38(3):106–117, 2018. doi: 10.1109/MCG.2017.3301120

[13] Y. Guo, H. Wang, Q. Hu, H. Liu, L. Liu, and M. Bennamoun. Deep Learning for 3D Point Clouds: A Survey. 8828(c):1–27, 2019. doi: 10.1109/TPAMI.2020.3005434

[14] J. Han, H. Zheng, Y. Xing, D. Z. Chen, C. Wang, and S. Member. V2V: A deep learning approach to variable-to-variable selection and translation for multivariate time-varying data.

[15] K. Hassani and M. Haley. Unsupervised multi-task feature learning on point clouds. *Proceedings of the IEEE International Conference on Computer Vision*, 2019-Octob:8159–8170, 2019. doi: 10.1109/ICCV.2019.00825

[16] C. Hermes, J. Einhaus, M. Hahn, C. Wöhler, and F. Kummert. Vehicle tracking and motion prediction in complex urban scenarios. In *2010 IEEE Intelligent Vehicles Symposium*, pp. 26–33. IEEE, 2010.

[17] P. Hermosilla, T. Ritschel, P. P. Vázquez, À. Vinacua, and T. Ropinski. Monte Carlo convolution for learning on non-uniformly sampled point clouds. *arXiv*, 37(6), 2018.

[18] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *32nd International Conference on Machine Learning, ICML 2015*, 1:448–456, 2015.

[19] G. Ji and H. Shen. Feature tracking using earth mover's distance and global optimization. *Pacific Graphics*, 2006.

[20] G. Ji, H. W. Shen, and R. Wenger. Volume Tracking Using Higher Dimensional Isosurfacing. *Proceedings of the IEEE Visualization Conference*, pp. 209–216, 2003. doi: 10.1109/visual.2003.1250374

[21] S. Lan, R. Yu, G. Yu, and L. S. Davis. Modeling local geometric structure of 3D point clouds using geo-CNN. *arXiv*, pp. 998–1008, 2018.

[22] Y. Li, R. Bu, M. Sun, W. Wu, X. Di, and B. Chen. PointCNN: Convolution on X-transformed points. *Advances in Neural Information Processing Systems*, 2018-Decem:820–830, 2018.

[23] L. Linsen, T. Van Long, P. Rosenthal, and S. Rosswog. Surface extraction from multi-field particle volume data using multi-dimensional cluster visualization. *IEEE Transactions on Visualization and Computer Graphics*, 14(6):1483–1490, 2008. doi: 10.1109/TVCG.2008.167

[24] T. Luciani, A. Burks, C. Sugiyama, J. Komperda, and G. E. Marai. Details-first, show context, overview last: Supporting exploration of viscous fingers in large-scale ensemble simulations. *IEEE Transactions on Visualization and Computer Graphics*, 25(1):1225–1235, 2019. doi: 10.1109/TVCG.2018.2864849

[25] J. Lukasczyk, G. Aldrich, M. Steptoe, G. Favelier, C. Gueunet, J. Tierny, R. Maciejewski, B. Hamann, and H. Leitte. Viscous Fingering: A Topological Visual Analytic Approach. *Applied Mechanics and Materials*, 869:9–19, 2017. doi: 10.4028/www.scientific.net/amm.869.9

[26] J. Lukasczyk, G. Weber, R. Maciejewski, C. Garth, and H. Leitte. Nested Tracking Graphs. *Computer Graphics Forum*, 36(3):12–22, 2017. doi: 10.1111/cgf.13164

[27] M. Monfort, T. Luciani, J. Komperda, B. Ziebart, F. Mashayek, and G. E. Marai. A deep learning approach to identifying shock locations in turbulent combustion tensor fields. *Mathematics and Visualization*, (9783319613574):375–392, 2017. doi: 10.1007/978-3-319-61358-1_16

[28] C. Muelder and K. L. Ma. Interactive feature extraction and tracking by utilizing region coherency. *IEEE Pacific Visualization Symposium, PacificVis 2009 - Proceedings*, pp. 17–24, 2009. doi: 10.1109/PACIFICVIS.2009.4906833

[29] V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Icml*, 2010.

[30] C. R. Qi, H. Su, K. Mo, and L. J. Guibas. PointNet: Deep learning on point sets for 3D classification and segmentation. *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, 2017-Janua:77–85, 2017. doi: 10.1109/CVPR.2017.16

[31] C. R. Qi, L. Yi, H. Su, and L. J. Guibas. PointNet++: Deep hierarchical feature learning on point sets in a metric space. *Advances in Neural Information Processing Systems*, 2017-Decem:5100–5109, 2017.

[32] T. Rapp, C. Peters, and C. Dachsbacher. Visual Analysis of Large Multivariate Scattered Data using Clustering and Probabilistic Summaries. *IEEE Transactions on Visualization and Computer Graphics*, pp. 1–1, 2020. doi: 10.1109/tvcg.2020.3030379

[33] R. Samtaney, D. Silver, N. Zabusky, and J. Cao. Visualizing Features and Tracking Their Evolution. *Computer*, 27(7):20–27, 1994. doi: 10.1109/2.299407

[34] F. Sauer and K. L. Ma. Spatio-temporal feature exploration in combined particle/volume reference frames. *IEEE Transactions on Visualization and Computer Graphics*, 23(6):1624–1635, 2017. doi: 10.1109/TVCG.2017.2674918

[35] F. Sauer, H. Yu, and K. L. Ma. Trajectory-based flow feature tracking in joint particle/volume datasets. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):2565–2574, 2014. doi: 10.1109/TVCG.2014.2346423

[36] K. Schatz, C. Müller, P. Gralka, M. Heinemann, A. Straub, C. Schulz, M. Braun, T. Rau, M. Becher, S. Frey, et al. 2019 ieee scientific visualization contest winner: Visual analysis of structure formation in cosmic evolution. *IEEE Computer Graphics and Applications*, 2020.

[37] D. Silver and X. Wang. Tracking scalar features in unstructured datasets. *Proceedings of the IEEE Visualization Conference*, 98:79–86, 1998. doi: 10.1109/visual.1998.745288

[38] H. Thomas, C. R. Qi, J.-E. Deschaud, B. Marcotegui, F. Goulette, and L. J. Guibas. KPConv: Flexible and Deformable Convolution for Point Clouds. 2019.

[39] F.-Y. Tzeng and K.-L. Ma. Intelligent feature extraction and tracking for visualizing large-scale 4d flow simulations. In *SC'05: Proceedings of the 2005 ACM/IEEE Conference on Supercomputing*, pp. 6–6. IEEE, 2005.

[40] 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), 2019. doi: 10.1145/3326362

[41] G. Weber, P.-T. Bremer, M. Day, J. Bell, and V. Pascucci. Feature tracking using reeb graphs. In *Topological Methods in Data Analysis and Visualization*, pp. 241–253. Springer, 2011.

[42] W. Wu, Z. Qi, and L. Fuxin. PointCONV: Deep convolutional networks on 3D point clouds. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2019-June:9613–9622, 2019. doi: 10.1109/CVPR.2019.00985

[43] J. Xu, S. Dutta, W. He, J. Moortgat, and H.-W. Shen. Geometry-Driven Detection, Tracking and Visual Analysis of Viscous and Gravitational Fingers. XX(X):1–18, 2019.