Fast and Robust Particle Data Feature Tracking Using Latent Vector Assisted Mean Shift

Category: Research

Paper Type: algorithm/technique

Abstract—Tracking features of time varying data is essential in scientific applications. However, most previous works on feature extraction and tracking focus on volumetric data, while a major part of scientific simulations generate particle data in the first place. Converting particles to volume will inevitably introduce errors and loss of details. On the other hand, considering the large number of the particles in most data sets, searching the entire data space in every time step for tracking is prohibitively expensive, which makes particle feature tracking a non-trivial task. In this work, a novel automatic particle feature tracking algorithm based on statistical feature definitions, mean-shift tracking, and neural networks is proposed. The temporal continuity of scientific phenomena allows us to search the neighborhood of particles between time steps, while the mean-shift method facilitates more efficient feature search. The representations extracted by an autoencoder neural network make the tracking more robust since it avoids the tracking processes stuck in local optimization points especially when the features are changing a lot between time steps. The effectiveness of our method is tested on several particle scientific data sets and the results were compared with other statistical feature tracking methods.

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

1 Introduction

When interpreting time-varying scientific data, it is essential for scientists to understand the change and development of features over time. Because of the complex nature of scientific phenomena, a precise definition of feature is not always available. Currently methods for identifying and tracking features fall into the following two categories:

- Algorithms for features that are well defined, and thus the feature extracting process is deterministic. There are a few previous works in this category. [5, 17, 20, 24, 25, 27, 31, 32]
- Algorithms [10, 18, 30] that allow feature extraction or tracking by statistical methods without deterministic feature definition.

Tracking algorithms for particle features, however, are scarce. In the context of scalar data visualization, feature tracking are mostly done through topological analysis that focuses on critical points, or geometric analysis that rely on the overlap of surface areas or volumes. Both of the methods, however, are not suitable for particle data, which is a primary data type for applications such as cosmology or molecular dynamics. The challenges come from the fact that topology for particle data are often not well defined, and the effectiveness of analyzing surface or volume overlap depends on how fast the feature moves. In addition, when successful tracking of features needs to consider additional physical attributes, topology or spatial proximity based methods cannot be easily generalized.

Mean-shift tracking [9] is a method where the histograms of target feature and candidate spatial region are first formed, and then the dissimilarity between the histograms of the target feature and that of the candidate region measured by Bhattacharyya coefficient is minimized. This method is built on the assumption that the Bhattacharyya coefficient forms a smooth terrain in the region of interest. This assumption may not hold true, however, if the region defining the feature is much smaller than the distance that the feature moves. In this case, we need information from a larger neighborhood to guide the movement of the feature window.

To achieve a more accurate, flexible, and faster temporal feature tracking algorithm for particle data, in this paper we present a latent vector guided mean-shift tracking algorithm. PointNet, a neural network based auto-encoder is used to assist this process by providing a latent representation of particle patches to assist the identification of features. PointNet is a deep neural networks which takes a point cloud as the input. Similar to the convolutional neural networks, The PointNet can work as encoder for an autoencoder. The idea of autoencoder can be dated back to 80s [11], which was traditionally used to perform dimensionality reduction and feature learning. This kind of autoencoders provides us with a more robust representation of features to guide the mean-shift algorithm. The contribution of the paper are as follows:

- We propose a feature tracking algorithm for particle data based on mean-shift tracking.
- We show that the representations of particle features can be well represented by a neural network based autoencoder called PointNet.
- We demonstrate that our tracking algorithm can achieve faster performance compared with more robust results.

2 Related Works

Our technique uses the PointNet autoencoder to assist mean-shift tracking for particle data sets. The literature that is related to our work includes scientific feature tracking, mean-shift tracking, and neural networks for point clouds.

Feature Tracking There have been plenty of works on volumetric feature extraction and tracking in the past. Earth mover's distance was used by Ji and Shen to design a tracking algorithm to find globally optimum feature [17]. In another work, Samataney et al. [24] proposed a approach based on volume correspondence to track scientific volumetric features. Chen et al. represents the tracked feature by feature trees in order to track features in distributed AMR data sets [5]. In a recent work, Schnorr et al. proposed a tracking method for sparse and space-filling data that determines the assignment of features between two successive time-steps by solving two graph optimization problems [26]. In a work from Sakurai et al., a systematically survey was done for frameworks designed for efficient development of feature tracking and feature definitions [23]. Compared to volume features, algorithms for tracking particle features are scarce. Sauer et al. utilized particle information from a joint particle/volume data set to achieve enhanced feature extraction and tracking [25]. One significant advantage of their method is that it allows to track features when sufficiently temporal sampling is not possible. .

Point Cloud Neural Networks The PointNet [21] is one of first methods in deep learning for point clouds 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 [22]. PointNet++ borrowed the ideas of

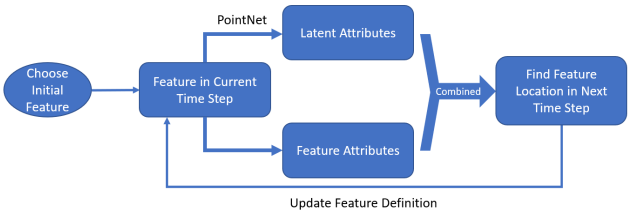

Fig. 1. Initial feature and definition attributes are chosen by domain experts. Feature attributes are original attributes defining the feature. Latent attributes is produced by PointNet Autoencoder giving particles positions and feature attributes. They are combined together to find feature in the next time step through mean-shift tracking.

hierarchically applying filters to extract both local and global features from Convolutional Neural Networks(CNN). Another effort to consider hierarchical information from point clouds is made by Zeng and Gevers [35]. They used KD-Trees to partition particles into hierarchical tree structure and learn the representation vectors progressively. There are other works focused on using point clouds and modifying the convolutional kernel to take point set input [29].

Neural Networks Application in Visulization With the booming of neural networks, more and more researchers from scientific visualization domain start to take advantage of it. However it is a significant challenge to exploit the immense amount of information that Deep Learning method can provide. One of the works that enlightened our work is from Cheng et al. [6]. They treat the neurons from a convolutional segmentation neural network as the latent representation for one specific voxel to assist hierarchically exploration of a volumetric dataset. There are other application of neural networks for volume rendering [3, 15] and super resolution [12] which also inspired our work.

Mean Shift Tracking Mean-shift was used as a real-time method to track non-rigid objects in video clips by comparing color histograms between frames. [9] It can achieve real-time performance with a high accuracy, thus is widely used in Computer Vision applications. Much efforts has been put to improve mean-shift tracking algorithms in the past. Yang et al. [4] improved the discriminality and reduced the complexity of computation by replacing Bhattacharyya coefficient with a new similarity measure in spatial-feature spaces. Collins [8] proposed a new mechanism to choose and update the mean-shift blob tracking kernel base on local maxima of differential scale-space filters. Dealing with the situation where multiple views of the target are available, Leichter et al. [19] enhances the Mean Shift tracker to use multiple reference histograms obtained from different target views. Another work by Ning et al. introduced Corrected Background-weighted Histogram (CBWH) to reduce background interference in target feature localisation.

There have also been mean-shift application for point clouds which is essentially equivalent to scientific particle data. Asvadi et al. [1] utilize mean-shift in 3D point cloud object tracking in the application of the perception system of autonomous cars. Mean-shift tracking was used on another 3D point cloud laser sensor data to track traffic participants by Hermes et al. [13]

In our work, mean-shift algorithm is applied on scientific particle data with high dimensional attributes.

3 Method Overview

Time-varying particle data are composed of consecutive time steps, each of which contains particles in 3-dimensional Euclidean space, and each particle can have physical attributes defined in an arbitrary number of dimensions. When the scientist spots a feature as a combination of particle positional and physical attributes in a particular time step, the goal of this work is to track the feature based on the attributes of particles in all the future time steps.

The workflow for our method is shown in Fig. 1. The features are first defined by domain experts through exploring the data set and circling a region of interest in the initial time step. The physical attributes that are relevant to the feature are also selected, hereafter called "feature attributes". The feature attributes and the distribution of particle positions contain most of the information that we need to identify the feature, except two deficiencies. Firstly, feature attributes are for individual particles without considering any correlations between the particles. Secondly, feature attributes may not have a smooth transition spatially, which is often required for robust feature tracking. To address these two issues, we propose to extract from the particles of feature a lower dimensional representation using the PointNet autoencoder. The PointNet autoencoder is trained to process the positional and feature attributes and produce more succinct information in the form of *latent attributes* to describe the feature.

The second part of our method is to track the feature between two consecutive time steps, which is achieved by the Mean-Shift tracking algorithm. The idea behind Mean-Shift tracking is to first represent the attributes of feature in the current time step as a histogram, and then to update the feature position in the next time step by finding the location that has the most similar histogram. In our case, the feature attributes and the latent attributes for particles in the neighborhood of the feature are combined together to form the histogram for Mean-Shift tracking. The detailed process of tracking is described in Sect. 5.

4 Attributes Encoding by PointNet

In this section we describe the autoencoder that is used to translate the attributes of particles around a specific feature position to a lower dimensional latent attribute. We call the group of particles input to the PointNet a particle patch. The latent attributes can make tracking of particle features more robust for three reasons: (1) Since the latent attributes are aggregated from a particle patch, they can better encode the relationship between neighboring particles. (2) Since the latent attributes are generated from a group of particles, they can describe the distribution of particles in the neighborhood better. (3) Since the particle patches used to generate the latent attributes as the feature moves have overlaps with each other, our generated latent attributes have a smooth spatial transition.

Autoencoder is an unsupervised artificial neural network designed to learn a representation for data set. It is composed of two parts: Encoder and Decoder. The encoder converts the original data to a latent representation, and the decoder converts this latent representation back to the data similar to the original one. The encoder and decoder can be any kind of neural networks. In our work, a PointNet is chosen as the encoder and the decoder is a fully connected neural network.

In our application, n 3D particles with a dimensional attributes around one specific position can be represented as a $n \times (3+a)$ dimensional vector originally, which can be quite large. Through our encoder, however, its latent representation can be reduced to less than 16 dimensions. Although it is inevitable to lose details after encoding the data with the autoencoder, this will not undermine our tracking since reconstructing the original data is not our goal.

4.1 PointNet as Encoder

The PointNet is designed to take points from an Euclidean space as input. Its pipeline is shown in the encoder part of Fig. 2, which is mainly composed of two parts: shared multilayper perceptron(MLP) and the max pooling layer. The MLP in the PointNet extracts the features from every point and then the max pooling layer aggregates the features from the points all together.

The MLP in our model contains 3 layers and each layer has 64, 128 and 1024 neurons respectively. The number of layers and neurons is chosen empirically balancing training time and reconstruction quality. The activation function after each layer is LeakyRELU [34], and the batch normalization layer is applied to improve training speed and stability of the encoder [16]. A max pooling layer is following the multilayer perceptron which converts n points each with 1024 channels into one vector with the length of 1024. This vector is then sent to a fully connective layer producing the latent vector whose length is of our choice.

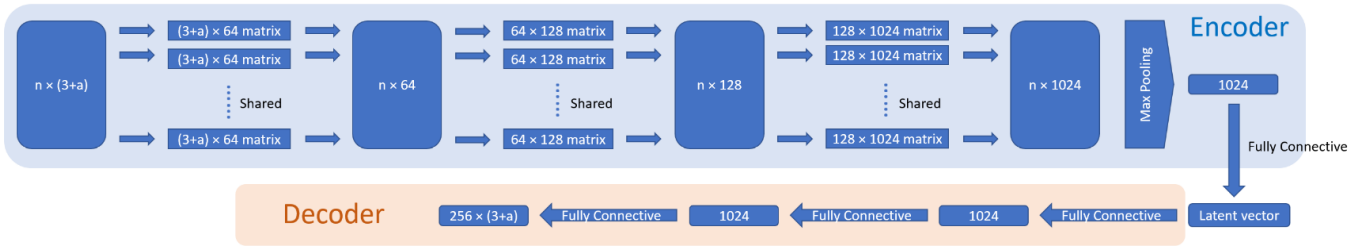

Fig. 2. PointNet is composed of several layers of shared multilayer perceptron (MLP) and a max pooling layer. The input to our network are n particles. Each of them has 3 dimensional coordinates and a dimensional feature attributes. Attributes and position of every particle are filtered by shared MLP and aggregated by max pooling. Every MLP is followed by Leaky ReLU activation and batch normalization. Decode is a fully connected neural network and its output size is fixed to $256 \times (3+a)$, which is rearranged to represent 256 particles' coordinates and attributes.

4.2 Decoder

The decoder is composed of 3 fully connective layers with 1024, 1024 and $256 \times (3+a)$ neurons, where a is the same with the number of input raw attributes. The activate function and batch normalization is set to be the same as the encoder. The last output of the fully connected layer is then reshaped to $256 \times (3+a)$, which can be treated as 256 points each with 3 dimensional position and a dimensional attributes. The number of reconstructed particles is not necessarily the same as the number of particles in the input, since the goal is to capture the feature of the point cloud but not the exact positions and attributes of them. Therefore the number of points in the output is fixed and chosen approximately to the input size.

Finally, the loss function is defined by the Chamfer Distance [2] between the reconstructed particles and the input particles. If we denote our input particles as U and the reconstructed particles as V. The Chamfer Distance from the point set U to V is defined as:

$$CD(U,V) = \frac{1}{n} \sum_{u_i \in U} \min_{v_i \in V} \|u_i - v_i\| \tag{1}$$

, where n is the number of points in U. Chamfer Distance does not require one-to-one point correspondence between the point sets, but captures the average mismatch from U to V. Because the Chamfer distance is not symmetric, the loss function is defined as $Loss = \frac{1}{2}(CD(U,V) + CD(V,U))$

4.3 Network Training and Testing

The input particles to the PointNet autoencoder are chosen from the time-varying data set. Because the size of particles from all time steps is too large, and our goal is to represent the particles in the neighborhood of a feature but not all the particles, particle patches are sampled from the data set as the input to the autorencoder. The patches are defined by the particles whose distance to a specific location is less than a radius r which is chosen based on the size of the data set. Also the particles' coordinates are translated relative to the sampled position. This way, the autoencoder will only learn the particle patches' shape and attributes but not their absolute positions in the domain. The sample locations can be chosen randomly or by some rules which are discussed below. The choice of r determines the neighborhood size of a feature. To make this query more efficient, data in every time step are first partitioned into a KD-tree, used for both training and testing.

K-Dimensional Partition The KD-tree is a tree structure whose leaf nodes contain raw particles and each branch represents a split in one dimension of the domain. KD-tree partition restricts the query of neighbouring particles to a subset of the data, which reduces the query complexity from $O(n)$ to $O(\log n)$, where n is the number of particles in one time step. The splitting of a dimension is done following the sliding mid-point rule. Since finding the median value is time consuming, this rule simply chooses the mid-range (the mid-way between the least and the greatest value) of a given dimension as the cut point, and to avoid producing empty leaf nodes, the split process ensures that each leaf node contains at least one particle. In addition, every time the dimension with the maximum range is selected to split until the number of particles in the leaf nodes drops below some preset threshold. The KD-tree is used in our work in the following 3 situations:

- Query particles in data patches of radius r in training and testing.
- Split the data by their attributes and get an unbiased patch sample.
- Query particles in radius h to define a feature in mean-shift tracking.

Besides this section, detailed applications of the cases are discussed in Sect. 5.

Dataset Bias How well the autoencoder assigns low-dimensional latent vector to represent a particle patch inputs depends on the training data we choose. Most scientific data have a large portion of nearly constant background without interesting features. If we randomly pick the sample locations, the sample attribute frequency will be uninteresting or the same in many particles. On the other hand, to get an unbiased training and validation data, we need to have the samples uniformly distributed in the attribute ranges. One way to get unbiased samples is to sample on the leaf nodes of attribute KD-tree.

To sample the attributes, another KD-tree is built for every time steps. Unlike the spatial KD-tree to query near particles, the split dimensions in this case are not positions of the particles but the attributes. Following the sliding mid-point cutting method, the KD-tree will evenly cut every dimension of particle attributes. Afterwards, sampling the same amount of patches in each leaf node will ensure generated training data are unbiased.

Inference and Latent Attributes After the autoencoder is trained using an unbiased dataset, it can be used to produce latent attributes for any given particle patch. As discussed earlier, the latent attributes represent neighborhood particles around the a given position. Consequently, in the inference phase, given a feature location, we can input the particles in the neighborhood to the autoencoder to obtain a latent representation.

5 Mean-Shift Tracking

Mean-Shift tracking is an efficient iterative algorithm to find the most similar features between two consecutive time steps by comparing the candidate locations and the target location's attribute distributions. In Sect. 4, we describe how to represent the particles' attributes with the latent attributes that capture succinctly the particles' positional and feature attributes in a small neighborhood. We have found that To have a complete representation of the particle feature, combining the latent attributes and the feature attributes will enable the mean-shift algorithm to perform more robust tracking.

In our method, a dimensional feature attributes and l dimensional latent attributes are concatenated together as the input to the mean-shift tracking algorithm. And our tracking goal between two consecutive time steps is to find the most similar candidate position in the latter

time step. Similarity is defined by Battacharyya coefficient between histograms of feature and candidate.

First we will introduce how to build our histograms. Assuming we are using particles centered at the location x_0 in a radius h to build their feature and latent attribute histogram $\mathbf{q}(x_0)$. To quickly obtain the particles in this area, the KD-tree by position discussed in Sect. 4.3 is used. The histogram is m dimensional with c bins in each dimension, therefore $\mathbf{q}(x_0) = \{q_u(x_0)\}_{u=1...c^m} (\sum_{u=1}^{c^m} q_u = 1)$. Let $\{x_i\}_{i=1,2...n}$ be the particles in consideration. We define a function $b : R^3 \rightarrow \{1...c^m\}$. This function associates the particle location x_i with a index $b(x_i)$ of the histogram bins where the particle attributes fall in. Also an Epanechnikov kernel is placed at x_0, giving less weight to particles on the edge of the feature neighborhood.

$$k(h) = \begin{cases} 1-h & if\ h < 1 \\ 0 & otherwise \end{cases} \tag{2}$$

Here h is particle's Euclidean distance to x_0. Our histogram bins are defined by:

$$q_u(x_0) = C \sum_{i=1}^{n} k(\frac{\sqrt{x_i^2 - x_0^2}}{h}) \delta[b(x_i) - u], \tag{3}$$

where δ is Kronecker function whose purpose is to filter out the particles that fall in bin u:

$$\delta(x) = \begin{cases} 1 & if\ x = 0 \\ 0 & otherwise \end{cases} \tag{4}$$

And C is the normalization term making this a probability density function.

Following the aforementioned equations we can calculate histograms $\mathbf{q}(x_0)$ and $\mathbf{q}(y_0)$ for the target feature in the earlier time frame centered at x_0 and candidate position in the next time step initially centered at $y_0 = x_0$, our mean-shift iteration will find the position by the weighted sum of the particles' coordinates around y_0. The weight for i^{th} particle are defined by the following equation:

$$w_i = \sum_{u=1}^{c^m} \delta[b(y_i) - u] \sqrt{\frac{q_u(x_0)}{q_u(y_0)}} \tag{5}$$

After having the weights, the next step is to move the candidate position to the weighted sum of particles to get the next candidate position $y_0^\star$:

$$y_0^\star = \frac{\sum_{i=1}^{n} y_i w_i}{\sum_{i=1}^{n} w_i} \tag{6}$$

Battacharyya Coefficient between $\mathbf{q}(x_0)$ and $\mathbf{q}(y_0)$ is defined as:

$$\rho[\mathbf{q}(x_0), \mathbf{q}(y_0)] = \sum_{u=1}^{c^m} \sqrt{q_u(x_0) q_u(y_0)} \tag{7}$$

If similarity increases at our new candidate position $y_0^\star$, we set $y_0 = y_0^\star$ and then calculate next $y_0^\star$ until the algorithm converges. Otherwise we shrink the update step by half $y_0^\star = \frac{1}{2}(y_0 + y_0^\star)$ until similarity increases. The proof for convergence can be found in the original mean-shift paper [9]. And detailed process is shown in Algorithm 1.

Algorithm 1 Mean-shift Tracking

```
Target feature predefined by the user.
Calculate the histogram q(x_0) for the target feature.
Initialize the candidate location y_0 in the current time step.
repeat
    Build histogram q(y_0)
    ρ[p(y_0), q(x_0)] = Σ_{u=1}^{c^m} sqrt(p_u(y_0) q_u(x_0))
    Calculate the weights w_i (i = 1...n) using equation (5)
    y_0^⋆ = (Σ_{i=1}^{n} y_i w_i) / (Σ_{i=1}^{n} w_i)
    ρ[p(y_0^⋆), q(x_0)] = Σ_{u=1}^{c^m} sqrt(p_u(y_0^⋆) q_u(x_0))
    while ρ[p(y_0), q(x_0)] < ρ[p(y_0^⋆), q(x_0)] do
        y_0^⋆ = (1/2)(y_0 + y_0^⋆)
        ρ[p(y_0^⋆), q(x_0)] = Σ_{u=1}^{c^m} sqrt(p_u(y_0^⋆) q_u(x_0))
    end while
    set y_0 = y_0^⋆
until ||y_0^⋆ − y_0|| < ε
```

6 Results

This section demonstrates how our proposed tracking algorithm works on two data sets. Our PointNet model is trained on a Super Computer with Intel Xeon 6148 and NVIDIA V100 "Volta" GPU and a Windows machine with Ryzen 2700X, 32GB of RAM and an NVIDIA RTX 2060 GPU. The tracking experiments are done on the Windows machine. PointNet autoencoder is implemented using Pytorch framework and mean-shift algorithm is implemented in Python.

6.1 Case Study 1: Salt Dissolution

The first case study is on a salt dissolution data set which is a time varying data with 50 different simulation runs each having 120 time steps. 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 tracking goal is to find the "finger" structures with high salt concentration, thus our experiments are merely done using the attribute concentration but not velocity.

6.1.1 Autoencoder Training and Reconstruction

This section shows the reconstruction quality of our autoencoder. All simulation runs and time steps were separated by the attributes into a KD-tree as discussed in Sect. 4.3. 60,000 positions were sampled from the leaf nodes of this KD-tree. We excluded the particle patches whose distance to the edge of the data set is less than their radius since there is not interested features in this area and cropped shapes of these patches will introduce unnecessary confusion to the autoencoder. 50,000 of them were used for training and the remaining 10,000 for validation. In our experiment, the latent vector length was set to 2 and 12 in 2 different models and the patch radius was chosen as 0.7 units which is of a similar scale as the interested features in this data set. The tracking results for 2 different models are shown in Sect. 6.1.2. After 50 epochs of training with the batch size equal to 32, the mean Chamfer distance between the original point cloud and the reconstructed point cloud converged at 0.037 and 0.031 for 2 models on the validation data set.

A reasonable reconstruction of the point clouds is another sign that our autoencoder is well trained. Our reconstruction results are shown in Fig. 4. It can be seen that the high concentration area in the input particle patch is captured by the autoencoder.

6.1.2 Tracking Accuracy

We now show the efficiency of our method by comparing it with direct applications of mean-shift tracking without latent attributes.

We chose the window size $h = 0.5$ when conducting the mean-shift tracking. This is approximately the size of the finger tip. The terminating threshold to the mean-shift tracking algorithm was set to 0.01 or 20 iterations, whichever came first. We controlled the number of bins to be the same when comparing our method with the direct application of mean-shift to ensure a fair comparison under the same amount of computation. Thus, we chose 1000 bins for the concentration attribute when directly applying mean-shift, and in our method 10 bins each for the concentration and 2 dimension latent attributes, which amounts to 1000 bins in total as well. The tracking result comparison is shown in Fig. 3. We compare the tracking result every 3 time steps. We can see the tracking window start to move off the finger tips between time step 13 and 16 when applying naive mean-shift tracking without

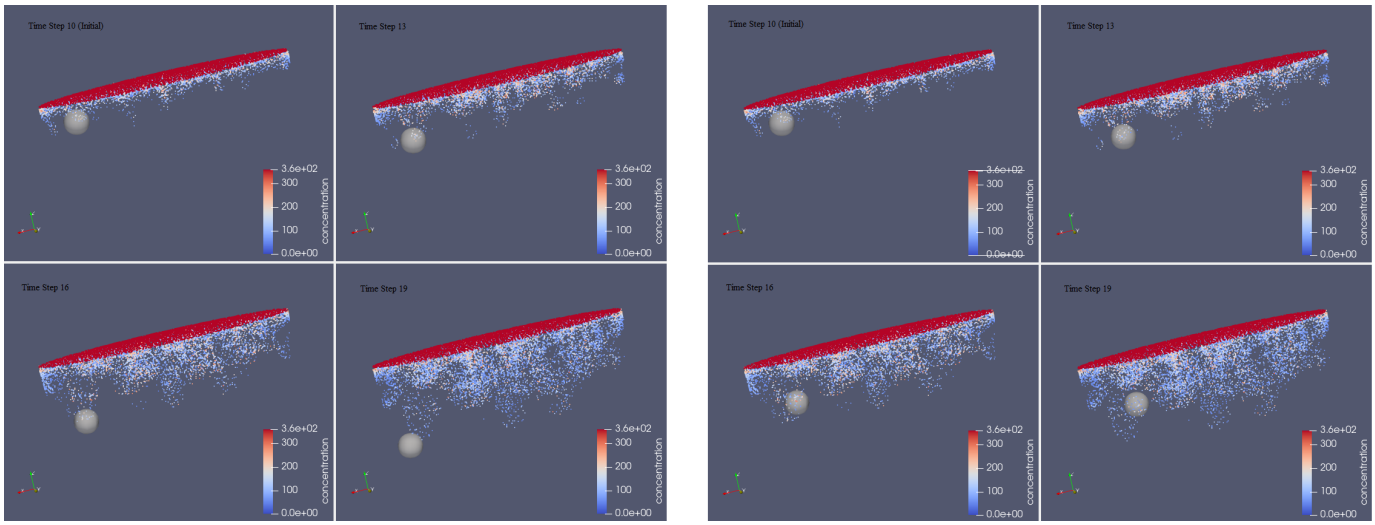

Fig. 3. Left: Tracking results with latent attributes. Right: Tracking results without latent attributes. Tracking is through 9 consecutive time steps. Initial feature "vicious finger tip" is chosen with the white sphere and feature candidates found in other time steps are also noted by the white spheres. This figure shows the tracking results in the time step 13, 16 and 19 from left to right and top to bottom.

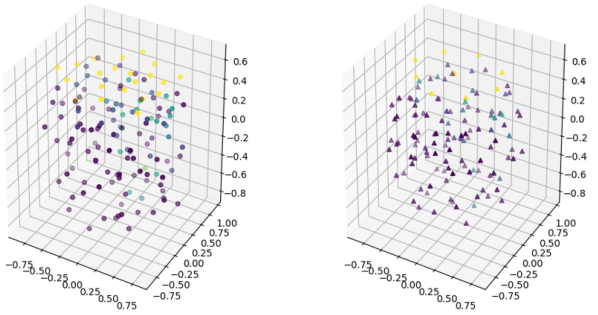

Fig. 4. Left: Input particles. Right: Reconstructed particles. Concentration is encoded by color.(Purple to yellow encodes lowest concentration to highest) Reconstruction captured high concentration on the top and relatively high concentration stripe across the point cloud body.

the latent attributes. However this did not happen till time step 19 for tracking using the latent attributes.

By calculating the Chamfer Distance between our tracked feature in every time step and the originally defined feature in the first time step, we can compare the accuracy of our method with naive mean-shift tracking without using the latent attributes. We also explore the influence of the number of latent dimensions on tracking accuracy. As shown in Fig. 5, three lines in each figure represent tracking without latent, tracking with 2 dimensional latent attributes and tracking with 12 dimensional latent attributes. The ensemble run and time steps of *Finger Tip 1* in the upper figure are the same with the result in Fig. 3 and the lower figure is the *Finger Tip 2* in another ensemble run and different time steps. Although it is inevitable that Chamfer Distance increases over time steps because of the deformation of the finger structure, tracking with latent attributes delays the significant distance increase and have an overall lower distance for all time steps in these 2 test cases. By randomly choosing finger tips to track, our method works better than naive mean-shift without latent in most cases and no worse in the rest. By comparing the tracking results from 2 dimensional and 12 dimensional latent, higher latent attribute dimension may bring up more accurate tracking on some harder tracking cases. However tracking with 12 dimensional latent takes 20+ seconds to run one time step and is nearly 15 times slower than 2 dimensional, even though we set number of bins in each dimension as 2 compared to 10 in the 2 dimensional case.

6.1.3 Visualize the Latent Attributes

Knowing what the latent vector encodes can help explain why mean-shift tracking becomes more robust with the latent attributes. To do this, we color the particles by their latent vectors as shown in Fig. 6.

Since the latent attributes we use have only 2 dimensions in one model, they are easy to visualize and explain. The first latent dimension has high values in and out of the finger area but low values on the edge of the finger tips. This works like a gradient filter to intensify the edge of our interested feature. The second latent dimension has high values in the area where the particles' feature attribute's value is high and low values where the feature attribute's value is low. The change of the value in this dimension is smoother across the edge compared to the feature attributes. This is similar to applying a low frequency filter to the original particles' attributes. The latent dimensions provide more information to better define the interested feature. Moreover these filters are automatically learnt by the neural network and do not require any user specification.

6.1.4 Mean-shift Optimization Landscape

Another way to explain the effectiveness of mean-shift tracking is to show the search space for feature similarity. The goal of mean-shift tracking is to find the most similar candidate by iterative updating the candidate position. As a result, we plot the similarity coefficient of all the possible candidate locations to compare our method and naive mean-shift. We use 2D latent vector model here for demonstration.

First, we sample the grid points around the initial feature position in the time step that we intend to locate the feature. Histograms are formed for every grid point and its Bhattacharyya similarity to the target feature histogram is calculated. In this way we can treat the result as a volume data with similarity as its attribute. Fig. 7 shows the volume rendering result. It can be seen that mean-shift with latent attributes greatly reduce the size of search space that has high similarity. Adding the latent attributes essentially gives a better definition of the feature itself, therefore greatly reduce the size of mean-shift search space.

6.2 Case Study 2: Dark Sky Simulation

Our second case study is using the cosmology simulation data set. This particle dataset is composed of 100 time steps with around 2,000,000 3-dimensional particles in each. The scale factor is different for every time step. However, after normalizing the particles' position, they are distributed in a $62.5 \times 62.5 \times 62.5$ Mpc/h space. Mpc/h is MegaParsecs

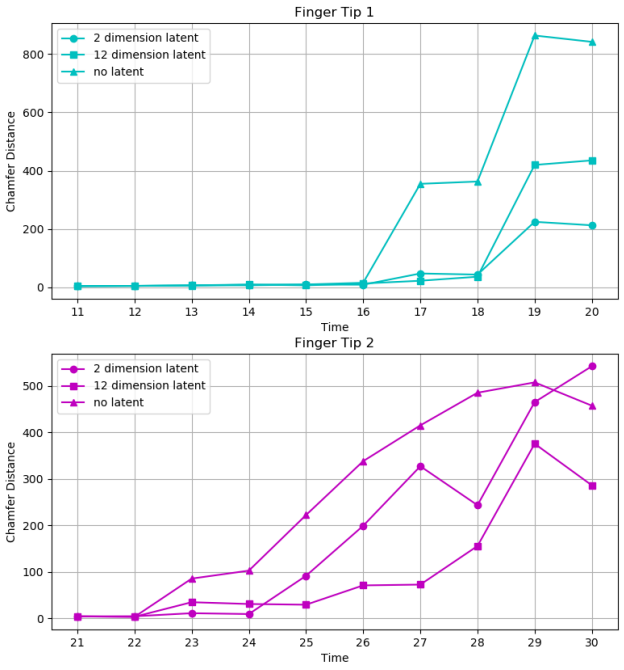

Fig. 5. Chamfer Distance between the feature in the initial time step and tracked feature in the current. Larger Chamfer Distance means lower similarity and lower tracking accuracy. Two Figures show the tracking results of two different Features. Circle denotes tracking with 2 dimensional latent, square dots denote tracking with 12 dimensional latent and triangle dots denote the tracking without latent.

divided by the Halo parameter, which is also the distance unit used in the provided Halo list [7]. Every particle has its position vector, velocity vector, acceleration vector, gravitational potential energy phi and a unique index. Along with these raw particle data, the data set comes with a Halo list containing *Halos*, which is a complicated feature defined by the particle density and other physical attributes in every time steps. 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 Halo moves between consecutive time steps.

The velocity vector, acceleration vector, and gravitational potential are chosen as our feature attributes since they are related to the Halo definition. Therefore, our feature attribute is in 7 dimensions. The dimensionality of the latent vector is chosen to be 6. Since the particle density in different areas of the data is quite different, if we sample the particle patch based on the distance to the patch center, distributions of particles in each patch will be quite different, making it hard for the autoencoder to capture the feature. Instead of using a spherical area, we choose 256 nearest neighbors as our particle patch. We sampled around 75,000 unbiased particle patches and used 50,000 in training and the rest 25,000 in validation. We terminated the training after 35 epochs when our loss value converged.

Our tracking window size is based on the actual Halo feature size and the termination criteria is the same as the previous Case Study, i.e., a threshold of the shift distance less than 0.01 or after 20 iterations. To have a fair comparison under the same amount computation. Our number of bins in the no latent attributes method is 4, and in the latent attribute case is 2. Since we have 7 dimensional feature attributes and 6 dimensional latent attributes, this gives $4^7 = 16384$ and $2^{13} = 8192$ bins under this 2 conditions respectively. Since we had the features' ground truth position, it is easy to compare the methods by calculating the distance from the track center produced by our algorithm and the real feature center. We randomly choose Halos from the data set Halo list and track them for 7 time steps. Error is recorded across 7 time steps for both our method and naive mean-shift. The result is shown

Table 1. Timing information under 4 settings for Case Study 2 in seconds. There is no total time and latent inference data for latent assisted mean-shift tracking without KD-Tree, for the experiment takes too long under this condition.

	Latent Assisted		No Latent	
	KD-Tree	No KD-Tree	KD-Tree	No KD-Tree
Total	20.67	*	1.84	0.92
Overhead	4.85	1.08	1.61	0.24
Histogram	7.07	7.13	0.19	0.67
Latent Inference	4.44	*	-	-

in Fig. 8. All three Halos have radius of 0.4 Mpc/h. Considering the feature radius, both 2 methods on Halo 1 is completed off the feature after tracking for 4 time steps. However our method's distance to the ground truth center is smaller than direct mean-shift in this case. Halo 2 and 3's results did not lose track of the ground truth feature till the last time step. And on these 2 cases our method is slightly better than naive mean-shift without latent attributes. Our tracking result for 30 particles starting from time step 12 till time step 99 is shown in Fig. 9. Most tracking results do not run off greatly from the Halo features.

6.3 Timing

In case study 1, our autoencoder takes 4 hours 27 minutes to train and validate for 50 epochs. This gives approximately 4 minutes to train and validate one epoch on NVIDIA V100. Our autoencoder for the second case study was trained on another Windows machine with NVIDIA RTX 2060. It takes 22 hours and 47 minutes to train 35 epochs and 36.5 minutes to train one epoch on average.

The running time for our algorithm is greatly influenced by the dimensionality of our latent and feature attributes and the number of particles in the feature window. For every particle in the feature window, we need to find its neighbor particles and do autoencoder inference to get the latent attribute. And the number of bins in the histogram is increasing exponentially with the latent and feature attribute dimension, so computational complexity is also increasing exponentially. There have been work to address this curse of dimensionality by Wei et al. [33] and their technique can also be adopted in our method.

In our first Case study it takes 0.75 seconds and 0.163 seconds to commit tracking between 2 time steps with and without latent attribute in average. The timing comparison for Case study 2 is shown in Table 1. We can see that using KD-Tree in our latent assisted tracking is crucial since during the latent inference, lots of k nearest neighbor query will be done. However in the naive mean-shift condition, introducing KD-Trees brings more overhead than the time it saves in knn query during histogram building.

7 Discussion

The above results demonstrate the effectiveness of our Latent Assisted tracking algorithm for tracking of data feature without a deterministic feature definition. Considering the non-deterministic natural of the feature, the way to evaluate our method is by statistical similarity and human knowledge. This is the situation in our first study case, where our algorithm was used in the setting of interactive exploration of an unfamiliar data set rather where the feature has no precise definition and deterministic location. In our second case study, we cross validated our method with a well defined Halo feature, showing that our method can successfully track the feature in most cases. In both cases the latent assisted method outperformed the direct application of mean-shift in terms of accuracy and effective tracking time range.

In the first case study, since our latent vector is only 2 dimensional, it is easier to give an explanation what the latent dimensions encode, as we have shown. However this kind of explanation is not always available if the latent dimensions increase and each dimension of the latent vector encode combined information. Unfortunately, it is inevitable to increase the latent dimensions when the data set becomes more complicated. Keeping in mind that our application scenario is mostly for data exploration, and tracking speed is a essential characteristics for

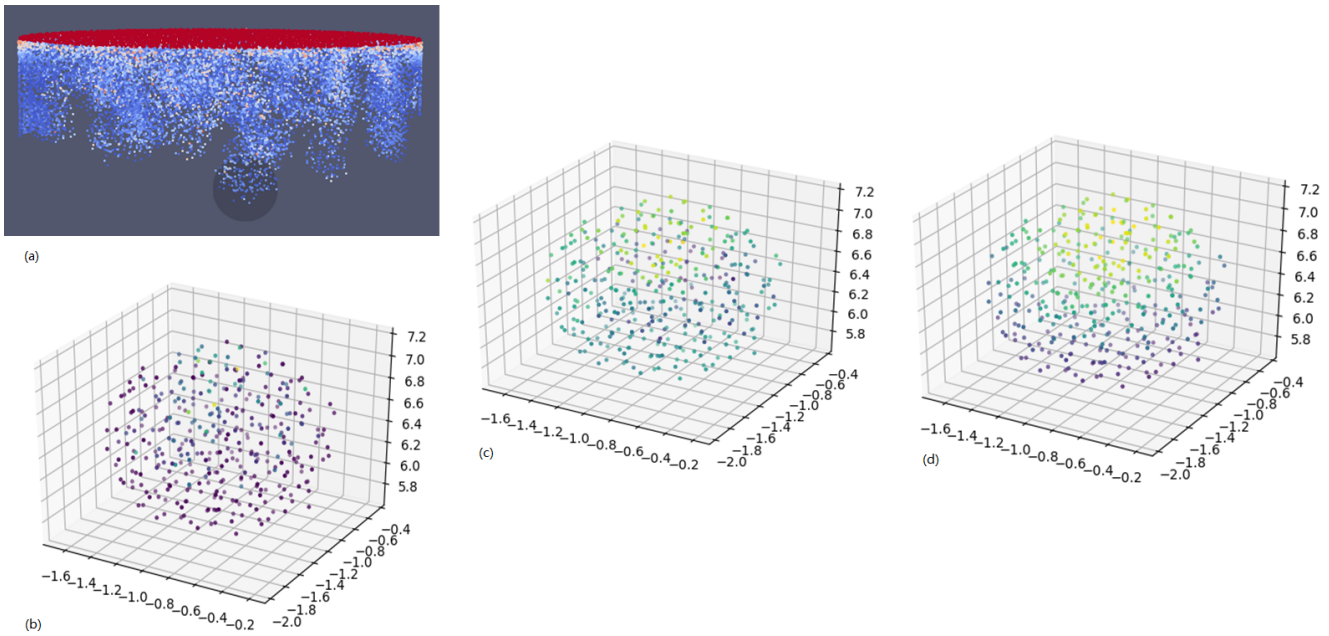

Fig. 6. (a) Interested feature area. (b) Feature area particles concentration. (c) Feature area particles 1st dimension latent vector. (d) Feature area particles 2nd dimension latent vector. It can be roughly seen that first latent dimension highlight the boundary area of the finger structure, which works like a gradient filter, and the second latent dimension is an average filter capturing the mean concentration around it.

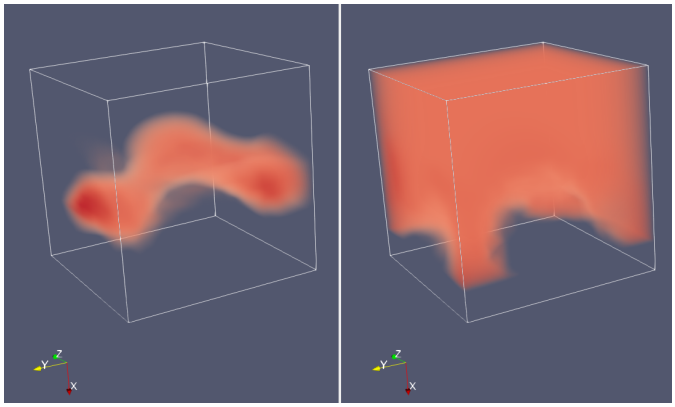

Fig. 7. Mean-shift similarity for our method (left) and direct application without latent vector (right). Darker red means higher similarity. Much more relative high similarity candidates can be seen in the right figure.

this. The limitations of using high dimensional latent attributes can be alleviated in several possible ways: (1) Applying dimension reduction techniques such as Principle Component Analysis to the latent space to generate a lower dimensional latent attributes or (2) training more explainable autoencoder with disentangled latent vectors like the one in *B*-VAE [14] and only select the most related dimensions for tracking.

Another factor greatly impact the performance of our method is finding the K nearest neighbour for latent inference. In our method KD-Tree is used to speed up the KNN query. However it is still possible to slow down our algorithm if the number of particles becomes too large. One way to solve this is subsampling the particles as used in the PointNet++ [22], however, it has the risk to lose feature details. The second way is to use activation maximization methods [28] to generate intermediate possible features between time steps to guide mean-shift tracking instead of directly adding latent attributes.

The overhead involved is one factor that needs to be considered when utilizing our algorithm. In addition to the KD-tree building overhead already discussed in Sect. 6.3, autoencoder training is another process which may take several hours to days depending on the machine in use before feature tracking can happen. However in real world applications, training can happen in-Situ along with simulations. This could be a potential solution to reduce the overhead.

8 Conclusion and Future Work

In this paper, we present a latent assisted particle feature tracking algorithm that does not require a precise feature definition. A PointNet autoencoder is trained on the tracking data set to represent particle clouds as latent vectors. The latent vectors are used as complimentary information in the tracking process. Our mean-shift tracking process locates the candidate feature position with the highest statistical similarity to the initially defined feature. Our method is evaluated on two different particle data sets and we also provide results to to explain why our method works. In the future, we would like to further accelerate our method and adapt it to more sparsely sampled time steps by replacing the latent inference in our method with the activation maximization techniques. Furthermore, we plan to develop a machine learning assisted feature exploration tool utilizing the proposed algorithm and integrating with an in-situ streaming data framework.

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] 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: Image Understanding Workshop*, pp. 21–27. Science Applications, Inc Arlington, VA, 1977.

[3] 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. doi: 10.1109/TVCG.2018.2816059

[4] Changjiang Yang, R. Duraiswami, and L. Davis. Efficient mean-shift tracking via a new similarity measure. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, vol. 1, pp. 176–183 vol. 1, 2005.

[5] J. Chen, D. Silver, and L. Jiang. The feature tree: Visualizing feature tracking in distributed AMR datasets. *PVG 2003, Proceedings - IEEE*

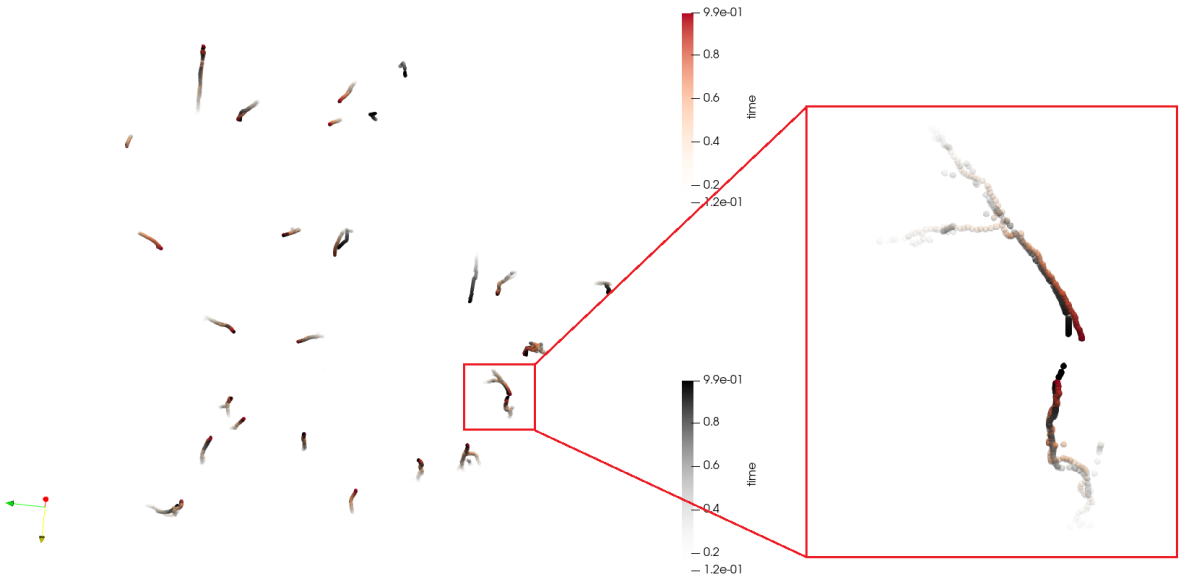

Fig. 8. Ground truth Halos are rendered with black spheres and our tracking results are rendered with red spheres. Opacity Encodes the time step and lowest opacity means the first time step 12. Note that the sphere radius does not encode the feature window size nor the Halo radius. It only denotes the center of tracked feature.

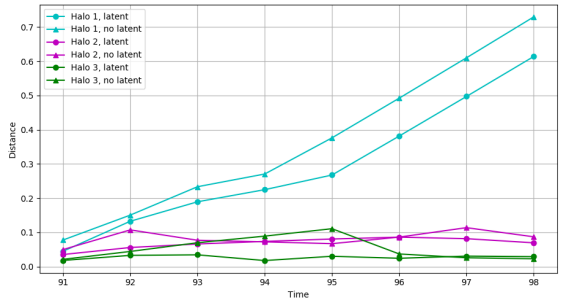

Fig. 9. Distance between the ground truth feature center and the center of tracking result from both latent assisted tracking and naive mean-shift tracking. Time steps are 91 to 99 and the 3 Halo features are randomly chosen from the list. Triangle dots represent tracking without latent and circle represent tracking with latent. Different color encodes different Halos.

Symposium on Parallel and Large-Data Visualization and Graphics 2003, pp. 103–110, 2003. doi: 10.1109/PVGS.2003.1249048

[6] 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

[7] 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

[8] R. T. Collins. Mean-shift blob tracking through scale space. In *2003 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2003. Proceedings.*, vol. 2, pp. II–234, 2003.

[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] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[12] J. Han and C. Wang. TSR-TVD: Temporal Super-Resolution for Time-Varying Data Analysis and Visualization. *IEEE Transactions on Visualization and Computer Graphics*, 2626(c):1–1, 2019. doi: 10.1109/tvcg.2019.2934255

[13] 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.

[14] I. Higgins, L. Matthey, A. Pal, C. Burgess, X. Glorot, M. Botvinick, S. Mohamed, and A. Lerchner. B-VAE: Learning basic visual concepts with a constrained variational framework. *5th International Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings*, pp. 1–13, 2019.

[15] F. Hong, C. Liu, and X. Yuan. DNN-VolVis: Interactive volume visualization supported by deep neural network. *IEEE Pacific Visualization Symposium*, 2019-April:282–291, 2019. doi: 10.1109/PacificVis.2019.00041

[16] 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.

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

[18] T.-Y. Lee and H.-W. Shen. Visualizing time-varying features with tac-based distance fields. In *2009 IEEE Pacific Visualization Symposium*, pp. 1–8. IEEE, 2009.

[19] I. Leichter, M. Lindenbaum, and E. Rivlin. Mean shift tracking with multiple reference color histograms. *Computer Vision and Image Understanding*, 114(3):400–408, 2010.

[20] S. Ozer, D. Silver, K. Bemis, P. Martin, and J. Takle. Activity detection for scientific visualization. *1st IEEE Symposium on Large-Scale Data Analysis and Visualization 2011, LDAV 2011 - Proceedings*, pp. 117–118, 2011. doi: 10.1109/LDAV.2011.6092327

[21] 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

[22] 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.

[23] D. Sakurai, H. Hege, A. Kuhn, H. Rust, B. Kern, and T. Breitkopf. An application-oriented framework for feature tracking in atmospheric sci-

ences. In *2017 IEEE 7th Symposium on Large Data Analysis and Visualization (LDAV)*, pp. 96–97, 2017.

[24] 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

[25] 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

[26] A. Schnorr, D. N. Helmrich, D. Denker, T. Kuhlen, and B. Hentschel. Feature tracking by two-step optimization. *IEEE Transactions on Visualization and Computer Graphics*, pp. 1–1, 2018.

[27] 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

[28] K. Simonyan, A. Vedaldi, and A. Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv preprint arXiv:1312.6034*, 2013.

[29] 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.

[30] 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.

[31] Y. Wang, H. Yu, and K. Ma. Scalable Parallel Feature Extraction and Tracking for Large Time-varying 3D Volume Data. *Eurographics Symposium on Parallel Graphics and Visualization*, D:55–62, 2013. doi: 10.2312/EGPGV/EGPGV13/017-024

[32] 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.

[33] T. H. Wei, C. M. Chen, J. Woodring, H. Zhang, and H. W. Shen. Efficient distribution-based feature search in multi-field datasets. *IEEE Pacific Visualization Symposium*, pp. 121–130, 2017. doi: 10.1109/PACIFICVIS.2017.8031586

[34] B. Xu, N. Wang, T. Chen, and M. Li. Empirical Evaluation of Rectified Activations in Convolutional Network. 2015.

[35] W. Zeng and T. Gevers. 3Dcontextnet: K-d tree guided hierarchical learning of point clouds using local and global contextual cues. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 11131 LNCS:314–330, 2019. doi: 10.1007/978-3-030-11015-4_24