Towards Object based Trajectory Representation and Analysis

Sameep Mehta, Raghu Machiraju, and Srinivasan Parthasarathy
Department of Computer Science and Engineering
The Ohio State University
{mehtas, raghu, srini}@cse.ohio-state.edu

1. ABSTRACT

In this article, we present trajectory representation and analysis algorithms for tangible object features found in temporally varying datasets. Rather than modeling the features as points, we take attributes like shape and extent of the feature into account. Our contention is that these attributes play an important role in understanding the temporal evolution and interactions among features. The proposed representation scheme is based on motion and shape parameters including linear velocity, angular velocity, etc. We use these parameters to segment the trajectory instead of relying on the geometry of the trajectory. Navigational and topological relationships can be readily derived from our representation. We also show how our scheme lends itself for predicting and understanding interactions among features. We evaluate our algorithms on real datasets originating from three different domains. We show the accuracy of the motion and shape parameter estimation by reconstructing trajectories with high accuracy. Finally, we discuss how the analysis coupled with domain knowledge can help in mining valuable information from the datasets.

2. INTRODUCTION AND MOTIVATION

Moving objects pose unique and exciting challenges in spatio-temporal data mining and databases. The key issues include representing, analyzing, and indexing the movement of the objects to gain better understanding about the evolution of an individual object and also to understand complex relationships among objects. The problem becomes even more challenging if the temporal behavior is characterized not only by the change in position of the object but also in terms of change in shape, size and object type. In this paper, we present such a representation scheme for object trajectories. We show how the representation can be used for establishing a variety of relationships among objects.

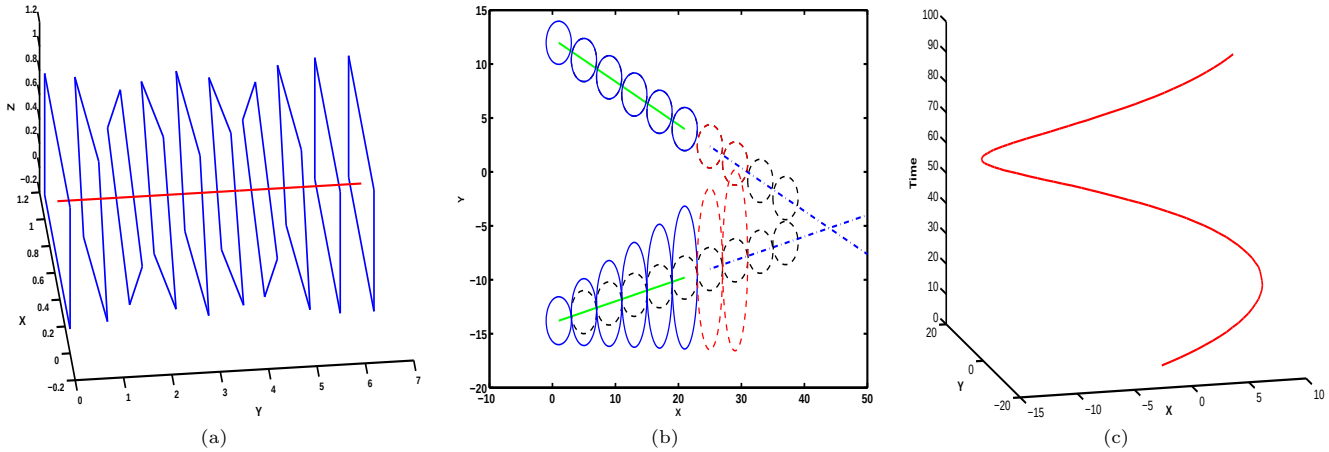
The main motivation behind our work comes from simple observations about objects and features[1] originating from scientific datasets. These objects have shape and size, i.e. they occupy some volume in space. Modeling these object with a single representative point, e.g. center of mass, amounts to loss of meaningful information. This abstraction can often result in misleading information about the motion characteristics of the object. Consider a simple example in Figure 1(a) where an object, is moving and rotating simultaneously along the y axis. The blue line shows the trajectory of the center of mass of the object. By analyzing the trajectory using a point based representation, we can only learn about the translational motion of the object. The information about the rotational component cannot be extracted. However, instead of a single point, if we record the position of K representative points of the object, along its surface, the presence of rotational motion can be detected, resulting in a better and more accurate description of the motion. In this paper, we refer to trajectory derived by monitoring a single representative point of the object as a *Point Trajectory* and the trajectory generated by taking in account the shape and extent is referred to as a *Object Trajectory*.

Another important characteristic of objects is that most of the time they interact exclusively with other objects in the same spatial neighborhood. To capture such interactions, correct neighborhood relationships should be established . To determine these relationships precisely, it is imperative that the size of the objects is taken into account for distance calculations. In this case also, a point based representation will lead to erroneous conclusions. Moreover, in scientific features, the extent of objects can also change over time. Ignoring this information can also result in an inaccurate and incomplete description of these relationships and will hamper the overall mining and prediction process.

For example, consider, Figure 1 which shows the point trajectory and object trajectory of two objects by green and blue color. The solid line represents the observed trajectory till time t. Initially, both the objects were circular however one object changes its shape to an ellipse. Suppose we want to find *at what time, will these object collide (merge)?* The dotted line shows the predicted path of the object by using the point trajectory. The line segments intersection corresponds to the meeting time of objects. The dotted black circles shows the path of the object without taking the shape change into account. Finally, the dotted red objects find the intersection point of the objects by considering the change in the shape. The shape and extent aware technique is clearly more accurate. We would also like to note that, even when the object boundaries are touching (and therefore, the objects are definitely interacting), the centers are quite far away (distance ~ 14 *units*). Therefore, using point trajectories, this interaction will either be missed or will be captured at a much later time.

Based on the above mentioned arguments, we believe that representation of a trajectory should consider the shape, extent and the change in the shape. Moreover, it should also consider description of the motion of an object. Porkaew et.al [25] categorized the trajectory representation in two

[1]Features are defined as Region of Interest (ROIs)in scientific datasets. In this paper we use the term feature and object interchangeably



(a) (b) (c)

Figure 1: (a)Point vs Object Trajectory (b) Effect of Shape change (c) Geometry vs Parameters

broad categories *Native Space (NS) and Parametric Space (PS)*. In *NS*, the d dimensional space is represented by a series of line segments or curves in $d+1$ dimensions, time being the extra dimension. In *PS*, the trajectory is represented by its motion parameters. Our representation includes the motion parameters. However, instead of concentrating only on linear velocity $\vec{v}$, we estimate angular velocity $\vec{\omega}$ and scaling co-efficient $\vec{s}$ as well. The choice of using parametric representation is primarily based on three reasons i) This representation provides a meaningful description of the motion as understood by humans, ii) it lends itself very well for analysis and prediction for spatial-temporal data and iii) the change in extent is much easier to account for in this representation than in the native space one. We derive the motion parameters by using the affine transformation matrix [22]. This matrix specifies the translation, rotation and scaling matrices. All the estimated parameters are collectively referred to as the **Motion Parameter Vector (MPV)**.

The movement of objects are based on laws of physics, resulting in smoothly varying trajectories which implies slow and gradual change in MPV. We take advantage of this property by segmenting the trajectory into sub-trajectories, such that the motion in each sub-trajectory can be represented by the same MPV. We believe that this segmentation is more meaningful than the segmentation based on geometry. Geometry based segmentation algorithms detect the change point by using different metrics, for example change in curvature. Consider the trajectory in Figure 1(c). Using geometry, we can find four piecewise smooth linear sub-trajectories (curves with degree 1). Increasing the degree to 2 will result in two sub-trajectories. However, by using the motion vector we can describe the whole trajectory by one MPV . One might argue that a higher degree polynomial could have approximated the trajectory in a better fashion using a single component. The argument is valid, however finding the right degree is often very difficult and is unlikely to be same across several trajectories. If polynomials of different degrees are used across different trajectories, then the trajectories cannot be compared with one another easily. The nice feature of our representation is that we don't have to change the definition of MPV based on the trajectory. For example if an object is not rotating, then the estimate for the angular velocity will be zero. Similarly for a rigid body (i.e. no scaling) the estimated scaling co-efficients will be again zero. Therefore, different trajectories can be compared using a single notion of similarity, although the weights across different elements of MPV will need to be determined.

Our final goal in proposing a new representation scheme is to understand the evolution of an object and its relationship with other objects present in spatio-temporal dataset. We identify a set of analysis strategies which help in attaining these goals. The strategies include deriving navigational and topological relationships using MPVs. Our representation can also be used for explaining and finding likely causes for critical events in scientific datasets. Finally, we show the usefulness of our MPVs for predictive analysis. To reiterate, the key contributions of this article are:

1. We develop algorithms for object based trajectory analysis. The features are modeled as objects with shape and extents as opposed to abstracting each object to a single point.
2. We present robust and intuitive methods for representing object trajectories by using important motion parameters. Additionally, we also capture the change in the extent of the objects.
3. We show how our representation lends itself towards understanding and predicting the evolutionary behavior of a objects. Additionally, complex relationships among objects can also be easily derived.
4. We empirically demonstrate the usefulness of our algorithms on datasets originating from three different domains. We also discuss the use of domain knowledge coupled in our algorithms for extracting useful information from these trajectories.

The outline of the paper is as follows: Section 3 presents basic problem definition, our parameter estimation and segmentation algorithms. Section 4 describes what kind of analysis can be performed based on our representation. Results on various datasets are provided in Section 5. In Section 6, we present a brief survey and highlight how our proposed methodology is related to existing literature. Finally, in Section 7 we comment on conditions under which our algorithms should be used and also outlines our ongoing and planned initiatives for this problem.

3. TRAJECTORY REPRESENTATION

The trajectory representation algorithms is divided into two steps. The first step takes the $3D$ objects as input and estimate the motion parameter vector (MPV). In the second step the MPVs are used to segment the trajectory into piecewise smooth sub-trajectories.

Basic Notation: S denotes a time varying dataset with N steps monitoring the movement of an object O, where O is represented by K points (landmarks) sampled from the surface of O [32, 26]. At i^{th} time step the state of the object in native space is represented by $O_i = [\{x_i^1, y_i^1, z_i^1\} \ldots, \{x_i^k, y_i^k, z_i^k\}]$. The position of j^{th} landmark at i^{th} time step is denoted by O_i^j. The time between two successive time steps is denoted by δ. In the parametric space the j^{th} sub-trajectory is denoted by O_j^p and is represented by the following feature vector

$$\{[t_1^j, t_2^j], [\{x_{t_1^j}^1, y_{t_1^j}^1, z_{t_1^j}^1\} \ldots, \{x_{t_1^j}^k, y_{t_1^j}^k, z_{t_1^j}^k\}], \{P_j^1, P_j^2 \ldots P_j^M\}\}$$

where $\{P\}$ represents the MPV of the j^{th} sub-trajectory. The time interval of j^{th} segment is $[t_1^j, t_2^j]$. The stored $\{x, y, z\}$ points describe the shape of the object at start of the segment i.e at time t_1^j.

Object Trajectory in Native Space: The trajectory in native space is obtained by concatenating the object representation for all time steps i.e $[O_1, O_2 \ldots, O_N]$.

Object Trajectory in Parametric Space: The trajectory in parametric space is obtained by concatenating the feature vector for all sub-trajectories i.e. $[O_1^p, O_2^p \ldots, O_J^p]$, where J represents the total number of sub-trajectories obtained after the segmentation algorithm.

3.1 Transformation and Physical Parameters

In this section we present the algorithm for finding physical parameters given a object trajectory in native space. The change in position and orientation of the object can be characterized by its linear velocity $\vec{v}$ and angular velocity $\vec{\omega}$. These parameters suffice when the size of the object is constant during its lifetime. However, this assumption does not often hold for real data. To account for this, we also compute scaling vector $\vec{s}$ of the object. For 3 dimensional objects, all size of all three parameters is 3×1. Let O_t and O_{t+1} be representation of object O at t^{th} and $(t+1)^{th}$ time step. O_t and O_{t+1} are related as :

$$O_{i+1} = S * R * O_i + T \tag{1}$$

where S is 3×3 diagonal matrix with scale co-efficients on diagonal, R is a 3×3 rotation matrix and T is 3×1 translation matrix. $\vec{v}$ is calculated by differentiating the T w.r.t time i.e. $\frac{\partial T}{\partial \delta}$. Similarly, the angular velocity is calculated by following equation $\vec{\omega} = \frac{\partial R}{\partial \delta} * R^{-1}$. The scaling vector $\vec{s}$ is specified by the diagonal entries of S. The Motion Parameter Vector (MPV) which describes the motion between between i^{th} and $(i+1)^{th}$ time step is denoted by $P_i = [\vec{v}, \ \vec{\omega}, \ \vec{s}]$.

If $[\hat{R}, \hat{T}, \hat{S}]$ are the best estimates of R, T and S respectively, then the reconstruction error [8] is given by the following error function:

$$e_{i,i+1}^2 = \sum_{j=1}^{K} ||O_{i+1}^j - (\hat{S} * \hat{R} * O_i^j + \hat{T})||^2 \tag{2}$$

The error between any two time steps t_p and t_q where $1 \leq p \leq q \leq N$ can be obtained by adding (integrating in continuous case) all the individual errors. The combined error function $E_{p,q}$ is denoted as:

$$E_{p,q}^2 = \sum_{t=t_p}^{t_q} e_{t,t+1}^2 \tag{3}$$

The optimal values of $\hat{R}$, $\hat{T}$ and $\hat{S}$ are those which minimizes the error function:

$$\min_{\hat{R},\hat{T},\hat{S}} \quad E_{p,q}^2 \tag{4}$$

Once we have estimated the matrices, the motion parameters can be calculated as follows: we set $q = p + 1$ i.e. we find the parameters between every two consecutive time steps resulting in a $(N-1) \times d$ parameter matrix P, where d is the length of each MPV. If we set $p = 1$ and $q = p + N$, then the method reduces to finding a single set of parameters, for the whole trajectory. Similarly, fixing $q = p + L$, is same as segmenting the trajectory in $\frac{N}{L}$ sub-trajectories and estimating a single set of parameter for each sub-trajectory.

We use the well known Levenberg-Marquardt optimization [20] for estimating the parameters which minimizes the above mentioned error function. The Levenberg-Marquardt method is a standard technique for nonlinear optimization, which employs gradient descent to search for minima of the error function. The initial condition is set randomly and number of maximum steps allowed is set high, when the function is invoked for the first time. The function returns the estimated parameter set P_1. For the second function invocation, instead of randomly setting the initial conditions we use P_1 and also decrease the number of maximum allowed steps to reduce the computational time. This strategy relies on the assumption that for a object moving under physical laws, the change in the motion is smooth if no external force is applied. Therefore we expect to find P_2 close to P_1. If the data follows a Gaussian distribution, then the optimal set of parameters can be estimated by using Singular Value Decomposition [31].

3.2 Segmentation

So far, we have obtained the MPV P_i thats maps O_i to O_{i+1} $\forall$ i $\in [1, N-1]$. Next, similar MPVs are clustered together to obtain sub-trajectories. We use a distance based clustering algorithm. We impose a constraint in the clustering algorithm so that each segment is continuous in time i.e P_i is added to j^{th} cluster iff the last MPV added to the cluster was P_{i-1}. A segment of length M is represented by ($ShapeDescriptor$, $Parameters$, T_{Start}, T_{End}) such that $T_{Start} - T_{End} = M$ and $T_{Start} < T_{Start+1} < \ldots < T_{Start+M}$

Distance Function: The MPV P_i can be thought of as a point in d dimensional space ($d = 9$ for three dimensional objects and $d = 6$ for two dimensional object). Instead of using the Euclidean distance metric over the whole space,

we can calculate the euclidean distance in subspaces (linear velocity ($\vec{v}$), angular velocity ($\vec{\omega}$) and scaling ($\vec{s}$) and combine these distances to obtain the final distance between two points. If $E(A, B)$ represents the euclidean distance between d dimensional points A and B, the total distance D between P_i and P_{i+1} can be calculated as

$$D(P_i, P_{i+1}) = E(\vec{v_i}, \vec{v_{i+1}}) + E(\vec{\omega_i}, \vec{\omega_{i+1}}) + E(\vec{s_i}, \vec{s_{i+1}}) \quad (5)$$

However, this formulation gives equal importance to all d dimensions which might produce misrepresented clusters. Consider a case where the object is rotating very slowly but moving very fast, in such a scenario we would like to assign more weight to the $\vec{v}$ than to $\vec{\omega}$. Similarly, an object can have a large velocity component in the x direction but moves very slowly in other directions. Therefore, we use a d dimensional importance vector I in distance calculations. The j^{th} entry of I specifies the weight assigned to j^{th} dimension. The weighted distance $W(A, B)$ between two d dimensional point is given by:

$$W(A, B) = \sqrt{\sum_{i=1}^{d} I_i * (A_i - B_i)^2} \quad (6)$$

Finding I manually can be cumbersome and error-prone. Therefore, we present an technique which automatically finds I. The idea is intuitively motivated by the fact that we want to assign more importance to fast changing parameters. Therefore, we assign I_i a value which is directly proportional to the variance in the i^{th} dimension. Let $V = (\sigma_1^2, \sigma_2^2, \ldots, \sigma_d^2)$ be the variance in each of the d dimensions (calculated over whole trajectory). I_i is calculated as $\frac{V(i)}{\sum_{j=1}^{d} V(j)}$. Thus the distance D between P_i and P_{i+1} is calculated as:

$$D(P_i, P_{i+1}) = W(\vec{v_i}, \vec{v_{i+1}}) + W(\vec{\omega_i}, \vec{\omega_{i+1}}) + W(\vec{s_i}, \vec{s_{i+1}}) \quad (7)$$

Segmentation Algorithm: Figure 2 shows the pseudo code for the segmentation algorithm. The input is the parameter matrix P of size $(N-1) * d$ and P_i denotes the i^{th} row of the matrix. ϵ specifies the error threshold. The first segment is initialized by the first MPV and the distance from next MPV is calculated using the distance metric discussed above. If the distance is $\leq \epsilon$, then length of the segment is increased by 1 and the average value of the parameters is calculated and stored. If distance is $\geq \epsilon$, then a new segment is initialized. Now, the next MPV can only be added to this new segment, thereby maintaining the temporal continuity in each sub-trajectory.

In this section, we described the algorithms for extracting the physical parameters from the $3D$ object trajectories and the associated representation scheme. Then, we presented the segmentation algorithm which divides the trajectory into smaller piecewise smooth sub-trajectories. In next section, we present the trajectory analysis techniques.

```
input: (P, ε)
out: Seg

int num = 1
Seg[num].left = 1
Seg[num].right = 1
Seg[num].param = P1
Seg[num].length = Seg[num].right - Seg[num].left

 for i: 2 to N-1
   if(Distance(Seg[numsegment], Pi) ≤ ε)
     Seg[num].right = i;
     Seg[num].param = (Seg[num].param*Seg[num].length+Pi) / (Seg[num].length+1)
     Seg[num].length++;
   else
      num++;
     Seg[num].left = i;
     Seg[num].right = i;
    Seg[num].param = Pi;
     Seg[num].length = 1;
   end if
 end for
```

Figure 2: Pseudo code for Segmentation algorithm

4. TRAJECTORY ANALYSIS

A multidimensional feature vector is generated and stored for each sub-trajectory. The feature vector includes $(Shapedescriptor, Object_{id}, MPV, T_{start}, T_{end})$. The shape descriptor stores the information about the shape at T_{start}. Apart from feature vector and shape descriptor, we also store other domain dependent information. Given an object id O and time interval $[T1, T2]$, we select all the sub-trajectories of object O whose time interval overlaps with $[T1, T2]$. Once we have all the sub-trajectories, we can use our analysis tools to understand the evolutionary behavior of one object and also the complex relationships among objects.

The techniques help to identify important objects and time instants. The definition of "important" is domain dependent. For example, a baseball moving with very high angular velocity may not be surprising however a defect structure in molecular dynamics with the same characteristics may point to some important physical process. Additionally, we also want to understand the process which triggered a critical event. Silver and Wang [28] suggested five basic critical events to understand evolutionary behavior of objects. The events are *Creation*, *Dissipation*, *Merging*, *Bifurcation* and *Continuation*. *Creation* event at $t = i$ refers to formation of a new feature which was not observed at $t = i - 1$. Similarly, *dissipation* event implies that the feature cease to exist $t = i + 1$. *Merging* event occurs when two or more features join. *Bifurcation* is exactly opposite of *merging*. Finally, *continuation* implies no change in the object. To reiterate, the main motivation is to provide techniques which will make the search process more focused and analytical reasoning more efficient.

Navigational Analysis: This analysis aids in understanding the motion characteristics of a object. The useful information which can be derived and used in analysis process are:

- **N1:** *What was the average (maximum and minimum) of velocity (angular velocity, sub-trajectory length and size) of object A between time* $[T_m, T_n]$*?*

- **N2:** *What was the trend in speed (angular speed and size) of the object A between* $[T_m, T_n]$*?*
- **N3:** *Which object displayed most (least) changing motion(size) between* $[T_m, T_n]$*?*

Once we have selected the set of relevant sub-trajectories, answering **N1** is trivial since we store all the required quantities. **N2** can be answered by further investigation of all the sub-trajectories of A between $[T_m, T_n]$. The trend is calculated by checking if the speed (magnitude of velocity) is decreasing or increasing. **N2** can help in understanding the evolution of the object. For example, a negative trend in the size will imply a shrinking object. Similarly, positive trend in speed will imply accelerating object.

N1 and N2 provides motion and extent statistics about a single object, whereas **N3** is designed to compare the motion of different object and select one with the desired property. **N3** extracts basic motion statistics (accessed via **N1**) for all objects and computes (weighted by lengths) variance for each object. Clearly, the object with lowest variance denotes least changing object. For example, if object A was represented by only one sub-trajectory, then $\sigma_A^2 = 0$. This analysis aids in identification of meta stable objects, which can then be further studied by domain experts. Please note that this analysis may look very simple, but the simplicity arise from our representation scheme. The same cannot be done in Native Space Representation (NSR) since no motion information is explicitly available.

Explanatory Analysis: This analysis tool aids in understanding complex relationship and to explain the critical events. We use our representation for understanding the following:

- **E1:** *Did object O crossed, entered or left a given area R between* $[T_m, T_n]$*?*
- **E2:** *What all objects are present in neighborhood of object O between* $[T_{c-h}, T_{c-1}]$*?*

One concern is that, all the above questions need the representation of the object in native space. However, we can very efficiently recover the native representation of the object using our feature vector. For each sub trajectory, the feature vector stores the native representation at the start of the sub-trajectory and the corresponding MPV. The MPV can, then, be applied to initial position to reconstruct the position at any time instant in that sub-trajectory. This reconstruction might induce some error. However since the trajectory segmentation algorithm is based on motion vectors, the reconstruction error is expected to be small. The object O_1 at time m is denoted by $NS_{1,m}$ in native space.

E1 describes some simple topological queries. O_1 is said to have *entered* R if $NS_{1,m}$ is outside R and $NS_{1,n}$ is inside R. For this paper, we consider O to be inside R, if at least n of its landmark points are inside R, where $n \leq K$. The parameter n governs how much minimum overlap is needed for a object to be considered inside of R. If n is set to 1, this test is effectively the same as in case of a point trajectory. Similarly, if n is set to K, the overlap test reduces to containment test. More advance relationships as described by Egenhofer [7] and Papadimas [21] can be easily accommodated. An extremely useful variant of **E1** is: *Find R, such that no object entered R between* $[T_m, T_n]$*?*. Identification of such an R can help explain many properties. For example, in CFD such an R will represent an area where the velocity field is weaker or aligned differently than all other parts. Thus, any path through R requires more energy and hence will not be taken. However, finding such a R without constraints will be computationally prohibitive. Therefore, we use a simplified version of **E1**, where the size and orientation of R is pre-defined but the actual position is not. A sliding window approach is employed to find the position. We place R such that the one side of R aligns with the boundary of the grid. Next, the native space representation of all the objects is calculated at T_m. If any object overlaps with R, then R is shifted by n units and the same procedure is followed. If no object overlaps at T_m, then the positions for T_{m+1} is calculated. This process is repeated till either we find an R with which no object overlaps between $[T_m, T_n]$ or R cannot be shifted. The latter case implies absence of such an R, with which no object overlaps.

E2 is used to understand the process that triggered a critical event for O. T_c represents the time of critical event and $h \geq 1$ represent how much historical (past) data we want to analyze. The neighborhood is defined by a region R around O. In the first step the native space representation of each object is calculated at T_{c-h} and the overlap test with R is conducted. The object ids of all the objects which pass the overlap test is stored. This process is repeated till T_{c-1}. The stored information can be now used to explain the start of interactions and critical events. A increase (or decrease) in the number of neighbors of O can mark the beginning (or end) of interactions and can point to a potential cause of a critical event. **Note:** $E2$ can explain bifurcation, dissipation and continuation events. However, merging and creation cannot be explained. Both merging and creation events results in formation of a new object at T_c. Since this object was not present in $[T_{c-h}, T_{c-1}]$, the area R cannot be defined and thus the neighborhood test cannot be performed. We present a scheme to handle the merging event in Section 5 with the defect case study. We are currently investigating the strategies to handle the creation event.

Predictive Analysis Prediction truly exposes the usefulness of our representation. Assume we have a parametric representation of object O_1 between $[T_1, T_N]$? We consider two types of predictions:

- **P1:** *What is the expected position of* O_1 *at time* T_{N+D}*?*
- **P2:** *What objects can interact with* O_1 *in next D time steps?*

P1 can be answered simply by accessing the last sub-trajectory of the object and successively applying the corresponding motion vector on shape descriptors. This kind of analysis only make sense if there is only one object. In case of multiple objects, O_1 can actually, interact with other objects in D steps which will result in a trajectory different from our predicted one.

P2 handles the case when more than one object is present in the dataset. In this case, we predict the position of all the objects for 1 time step only. Next, **E2** is used to find if any object's predicted position is in O_1's neighborhood. If no such object is found, we can predict for the next step and repeat the procedure. This is continued, till we find a object in O_1's neighborhood or we have predicted the position for

D steps. If we find such a object in O_1 neighborhood, then we mark that object as a possibility for interaction.

The quality of prediction is highly dependent on the value of D. Assigning a very large value to D is a unrealistic and may not produce correct results. A simple upper bound on the value of D can be easily derived. Assume that a trajectory of length N is segmented into J segments. The expected length of each segment is $B = \frac{N}{J}$, which means on average the motion changes every B time steps, which implies that we can only predict upto $B-1$ time steps which acts as our upper bound. The parameter D can take the values between 1 and $B-1$. In most of the physically derived datasets, the change in motion is very smooth. Therefore, we can expect a very small value of J and therefore a reasonably high value for B.

Example: We present one example to demonstrate the usefulness of the proposed analysis on a vortex datasets *VD1*. The simulation starts with four vortices. In the course of simulation the vortices go through a number of changes including change in position, size, and number. Figure 3(a) shows the vortices (solid black objects) detected from *VD1* at $t = 40$. The green markers represent the position of the **reconstructed trajectory** after MPV estimation and segmentation. The area $R1$, represented by dotted boundary, is defined to show the use of **E1** and area $R2$, represented by solid boundary, is used to demonstrate the usefulness of **E2**. Figure 3(b) and (c) shows the snapshot of simulation at $t = 80$ and $t = 120$ respectively. Using **E1** we are able to capture the entrance of vortex $V1$ in $R1$ between time interval $[40, 120]$. *Even though $V1$ is moving out of $R1$, according to our inside-outside test, it is still considered inside.* Please note that if we change the time interval to $[40, 140]$, we can capture that $V1$ has crossed $R1$. Similarly, if time interval $[90, 140]$ is used we can capture that $V1$ left $R1$.

Figure 3(c) shows a bifurcation event where vortex $V2$ split into two smaller vortices. We employed **E2** to find a potential cause for this event, with $T_c = 120$ and $h = 70$. *Please recall h is used to specify the number of past time steps to be used.* We found that vortex $V4$ entered $R1$ at $t = 60$ which can point to the start of interaction among $V4$ and $V2$. At $T = 110$, the distance between the $V1$ and $V2$ grew smaller, which indicates stronger interactions. These interactions are the most likely cause of bifurcation event at $T = 120$. We would also like to point that at $T = 60$, even though the objects were interacting, the center of objects were quite far away.

Figure 3(a)-(c) also demonstrates the effectiveness of the parameters estimation algorithm. For example, the shape of $V1$ has changed considerably from Figure 3(a) to Figure 3(c). However, this change is captured effectively by scaling parameters as reflected by the positions of the reconstructed sample points (green markers).

Figure 3(d) shows the use of $P1$ and $P2$. **Due to space constraints, we are forced to present the prediction at two different times in the same figure. For $P1$ please pay attention only to the gray ellipse. For $P2$, consider rest of the plot.** We used $P1$ to predict the position of $V1$ at $t = 120$. Figure 3(c) shows the position of $V2$ as generated from the simulation. It is evident from the plots that the position and orientation was captured accurately. The overall change in shape is also captured very well. *Please note that right now, we only handle affine transformations, therefore, non-rigid deformations cannot be captured. This is one of the subject for further study.* Next, we used $P2$ for predicting interactions for $V2$. We found that $V4$ is expected to enter the neighborhood of $V2$ (defined by $R2$) at $t = 62$. In reality, $V4$ entered $R2$ at $t = 60$ which is very close the predicted value.

5. RESULTS

In previous section, we presented results on one vortex dataset. In this section, we present results on baseball datasets, another vortex dataset and a defect dataset.

5.1 Case Study 1 : Baseball Trajectories

We generated a collection of baseball trajectories based on the simulation model described in [1]. The model take into account air drag, gravity, initial velocity, angular velocity and angle of throw. Based on all these attributes we can generate different kinds of baseball trajectories namely, *slide ball, fast ball, curve ball* etc. This case study is very useful because it helps us to evaluate the quality of our parameter estimation and segmentation algorithm. We control the input parameters and we also have access to simulation parameters at every time step. Therefore, we can check the simulation parameters against the derived parameters for evaluating the quality of MPV estimation. Please note that the parameters from the simulation are not used in trajectory analysis algorithms. They are only used to compare the final results. For the other two case studies we don't have the luxury of knowing the actual motion parameters. At every time step we sample 20 points from the surface of the ball. These points are considered as the shape descriptors for the baseball.

Figure 4 shows the results of motion parameter estimation using four different techniques. Figure 4(a) shows the result when only point trajectory was considered. This method fails to capture the rotational aspect of motion. Figure 4(b)-(d) show the original trajectory with dotted lines and the re-constructed one with solid lines. The direction of the arrow (shown only on solid lines) shows the direction in which the ball is moving. Figure 4(b) shows the original and reconstructed trajectory using global fitting i.e. one set of motion parameters are estimated which fits whole of the original data optimally in least square sense. Figure 4(c) shows the result, when instead of using our segmentation algorithm, a fixed size segmentation (40 steps) is used i.e. a single $\vec{v}$ and $\vec{\omega}$ is estimated for every 40 time steps. In some parts of the trajectory, this scheme performs very well, however in other parts the performance is really bad. This behavior is not very surprising, if the segment boundaries align with the actual motion change boundary, the results are very good. However, if a motion changes within a segment, the error is high. Figure 4(d) shows the results of our algorithms.

For a simulation of $8,000$ time steps, we found 9 sub-trajectories. The root mean square error between original and re-constructed trajectory was 11.1, 4.8, and .83 for global, fixed segmentation and our approach respectively. The root mean square error between the actual and estimated parameters (averaged over all sub-trajectories) was 9.34, 3.86, and .51 for global, fixed segmentation and our approach respectively. The low RMSE error in both cases show our approach's effectiveness in capturing the motion parameters.

5.2 Case Study 2 : Vortex Trajectories

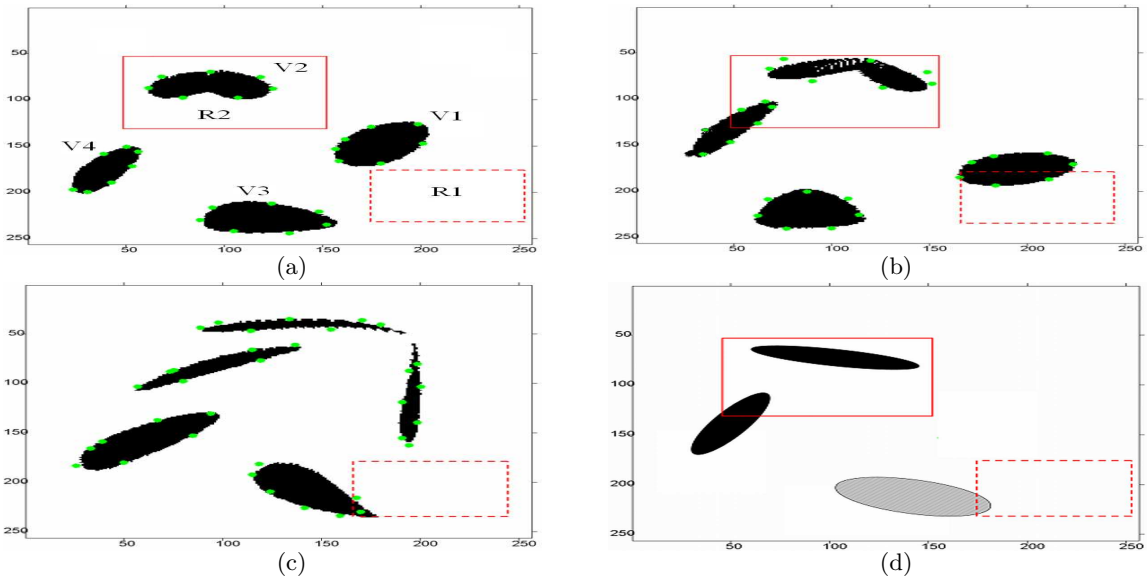


Figure 3: Vortex Datasets 1: (a) This dataset starts with 4 vortices (b) Start of interactions between vortices (c) Bifurcations Event (d) Predicted Positions and Interactions.

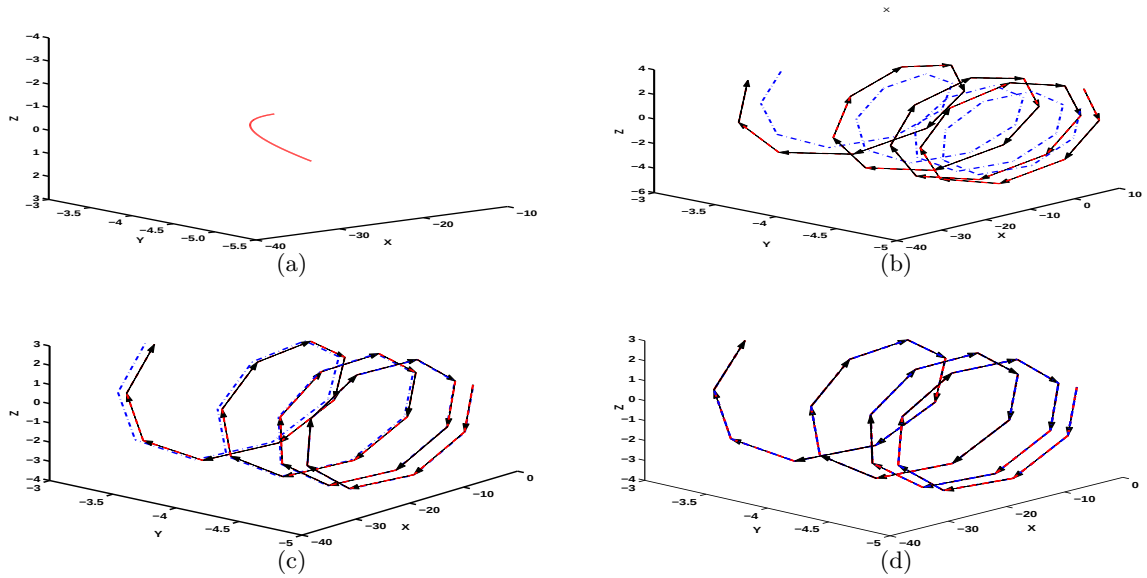


Figure 4: Baseball Trajectory (a) Using only center of mass (b) Global parameter estimation (c) Fixed length segmentation (d) Proposed approach.

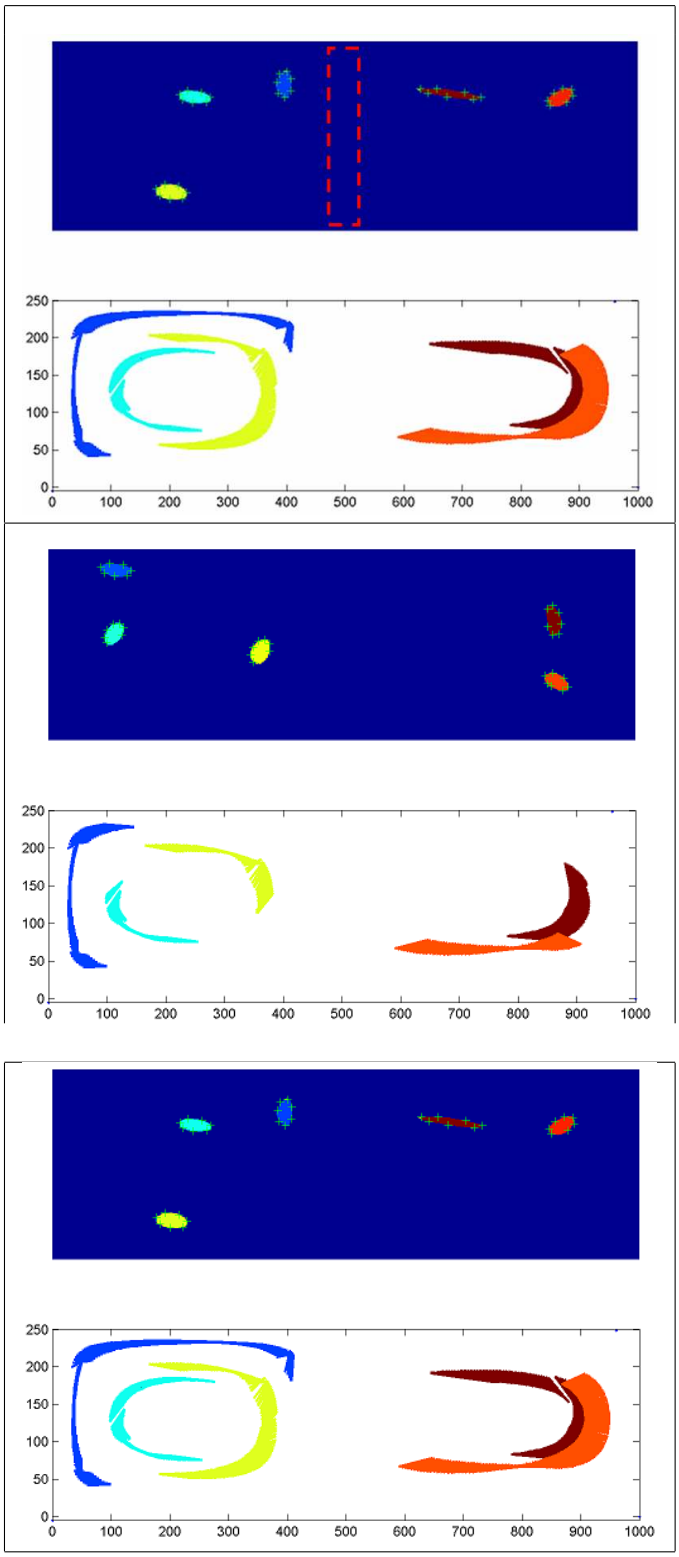


Figure 5: Vortex Dataset 2: Visualizing the vortex trajectories at different times of simulation

The vortex datasets was generated using the algorithm proposed by Kim et.al [13]. We use the vortex detection algorithm presented by Jiang et.al [11]. The shape of a vortex is approximated by an ellipse. Next, 10 points were sampled from the boundary of the ellipse. These sampled points are used to estimate the motion parameters between two vortices in consecutive time steps. Figure 5 shows the snap shots of simulation and results at different times. The topmost plot at each time shows the actual vortices and the green markers denote the sampled boundary points for the **reconstructed motion** at a given time instant. The first plot can be used to qualitatively evaluate our parameter estimation and segmentation algorithms. For a reconstructed trajectory with zero or negligible error, all the markers should lie on the actual object boundary (if object is a perfect ellipse) or very close to the object boundary otherwise. The lower plot shows the evolution of the object. The extent and orientation of the object is represented by the length and tilt of the corresponding major axis. *The minor axis is not shown for aesthetic reasons.* We have generated a movie file for both the vortex datasets. The files can be accessed at http://www.cse.ohio-state.edu/∼mehtas/Video/index.html. Using our analysis method $E1$ we were also able to find a rectangular region (shown by the dotted rectangle in the first plot), where no vortex entered during the whole simulation. Moreover, while analyzing the MPVs, we found that the three vortices to the left of the box always moved in clockwise fashion whereas the vortices on right always moved in counter-clockwise fashion. All these phenomenon can be easily seen in the video file.

5.3 Case Study 3 : Defect Trajectories

Our last dataset originate from Computational Molecular Dynamics (CMD). The features of interest in this domain are defect structures. We used the algorithms presented by Mehta et.al [17] for defect detection. The 8 corners of the 3d minimum bounding box of each defect are used as shape descriptors. Till now we have shown the examples of 3 types of critical events, *creation* and *bifurcation* in $V1$, *continuation* in $V1$ and $V2$. This specific datasets was selected to show our approaches to explain the merging event. This simulation starts with two separate defect structures which merge at 5000^{th} time step. Figure 6 shows the evolution of the defects. In the figure, we have projected the defects to xy plane. The dotted line shows the movement of center of mass of the defects. The solid line depicts the extent of the defects. We would like to point out that, even when the defect boundaries are touching (a precursor to merging event), the center of mass of the defects are far apart (3d distance=8 units). We note that, by using the point trajectories, we would not be able to find when exactly the event initiated.

The description of $E2$ given in Section 4 cannot handle merging events. We now briefly discuss the modification to $E2$ for explaining a merging event. Every merging event is accompanied by dissipation of two or more features. For example in Figure 6, after the two small defects merged to form a larger defect, both the smaller defects ceased to exist. Therefore, for explaining a merging event, we search the neighborhood of all the dissipated features starting at T_{c-h} till T_{c-1}. If two or more objects are detected in each other neighborhood, then it is highly likely that these objects merged to form the new defect. For example in Fig-

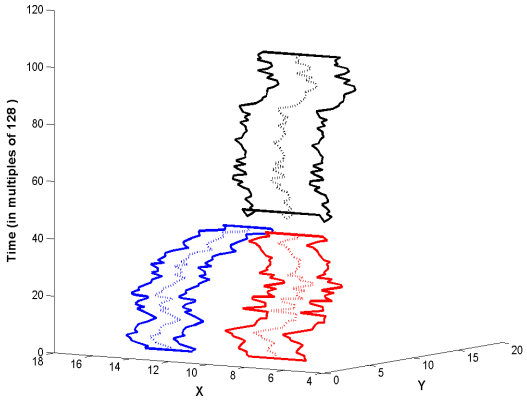


Figure 6: Defect Datasets: The figure shows the merging of two defects to form a new bigger defect.

ure 6, the two defects are in each other's neighborhood when there boundaries are touching. With this modified definition of $E2$, we were able to explain the merging event successfully. Previously, we also demonstrated the use of this approach in $2D$ case [33].

6. RELATED WORK

Our work is related to trajectory representation and spatial temporal query processing present in existing literature. First of all, we would like to note that there are significant differences between trajectory and time series data [6]. Techniques presented with respect to one dimensional time series like the ones based on DFT [3, 9], DWT [24], SVD [16] are not directly applicable in this context.

The trajectory representation can be broadly classified in two categories Native Space Representation (NSR) and Parametric Space Representation (PSR). Here we concentrate on the use of parametric space representation. Please see [18] for an excellent survey on most popular NSR techniques.

Cai et.al [5] proposed the use of Chevesheyv polynomials for global approximation of the trajectory. Recently Ni et.al [19] extended the idea which allows variable length trajectory and also facilities using different degree polynomials for different trajectories However, in both the efforts a d dimensional trajectory is treated as d separate one dimensional trajectories. Such an approach will lead to loss of interdependence among dimensions. For example in case of circular motion the position of x_{t+1} is given by $x_t * cos(\theta) + y_t * sin(\theta)$, which is dependent on y_t (value of y at previous time step). Moreover, in our case each object is represented by K points which will lead to $3 \times K$ time series. Chen et.al [6] and Bakalov et.al. [4] extended 1 dimensional Piecewise Aggregate Approximation [12] to handle d dimensional trajectory. The method partitions the space, assign a symbol to each grid cell and approximates the trajectory by the symbols of grid cells which contains some part of the trajectory.

Kollios et.al [14] represented the trajectory by its velocity and used kdB tree to index and process predictive queries. Saltenis et.al. [27] used a very similar representation and showed that PS indexing is very well suited for predicting the future position of the object. Tao et.al [30] presented TPR* tree which indexes linear velocities of the objects. The usefulness of storing linear velocity instead of actual object location is shown by better prediction and low overhead in terms of updates to the index structure. Porkaew et.al [25] compared the trajectory representation in Native and Parametric space for historical queries. All these methods divide the trajectory into segments with constant velocity. Agrawal et.al [2] presented a scheme to index trajectories with non-linear motion. However, the problem definition was different from ours. In that work, the authors assumed that the motion parameters are known in advance for all trajectories and concentrated in indexing. Recently Tao et.al. [29] presented a recursive function based algorithm for learning the motion type of an object. The algorithm define parameters: retrospect f and horizon to find the motion type. For, simple motions small f suffices however, for complicated motion types a larger retrospect is needed. Thus the choice of retrospect is very crucial and different trajectories may be represented by using different retrospect. For a d dimension object, the algorithm need to estimate $d^2 \times f$ parameters. The paper reports $f = 6$ as good choice for most motions in 2D i.e. $d = 2$. Assuming that $f = 6$ holds for higher dimensions also, then in best case scenario, $3^2 \times 6 = 54$ parameters are needed for a three dimensional trajectory. Finally, the quality of the results may degrade in the presence of noise. All the approached reviewed so far abstract the object by a point. Given the position of object at two successive time instants, a translation matrix (and hence linear velocity) which optimally maps one point to another can be derived. Estimation of both angular and linear velocity from two points is an ill-posed problem. Additionally, since only points are considered, scaling is not defined.

The most closely related work to the one presented in this paper is by Hadjieleftheriou et.al [10] and Kollios et.al [15]. The proposed solution take into account the change in extent of the object. The basic notion is to segment the d dimensional trajectory into smaller sub trajectories using the geometry. Then for each subcomponent, find d polynomial functions such that each one optimally fits one of the d dimensions. The whole trajectory can be described by collection of all these functions. At a conceptual level this work and ours are totally different. Foremost, the main focus of work described by the author in [10] is indexing and not motion description. No explicit modeling the velocities or scaling parameter was described. We highlighted some other differences in Figure 1(c) in Section 2. We again mention that treating each dimension of a trajectory as individual time series can results in unintuitive representation of the trajectory.

Recently, we [32, 33] presented algorithms for mining frequent spatial patterns from scientific datasets. The main goal of that work was to understand the evolution of spatial patterns and use that information to reason about the critical events. Study of the motion of individual objects was not performed. Additionally, *predictive*, *navigational*, *topological* and *interaction* analysis was not discussed in this previous work.

7. DISCUSSION AND CONCLUSIONS

In this article we presented algorithms for extracting meaningful and easy to understand representation of object trajectories. Instead of treating the object as a point, we take

into account the shape and extent of the object. We would like to point out that, our goal was not to develop generic time series analysis methods. Our algorithms are specifically geared towards understanding the motion behavior of physical objects. For such objects, the motion has been traditionally understood in terms of speed, and direction. For time series datasets like stock prices or sensor data, this scheme may not produce meaningful representation. We also capture the change in size of the object, which is very common in scientific datasets, by using scaling parameters. The algorithms performs extremely well on datasets originating from three different sources.

In this article, we only consider rigid transformations. Due to this constraint, we cannot fully quantify the change in shape of the objects. With scaling parameters we only capture the change in extent of the object. Currently, we are exploring algorithms for estimating non-rigid transformations like stress and strain. These parameters play an very important role in crack and fracture propagation in materials. In this paper we used R tree for storing and retrieving sub-trajectories. In the future, we would explore the use of other advanced and efficient spatial-temporal data structures like TPR [27], TPR* [30] and TB [23]. Currently, we are also developing a visualization framework to support interactive analysis and reasoning for the object trajectories through user interface.

8. ACKNOWLEDGMENTS

All authors contributed equally to the intellectual content of this article. This work is funded by the following NSF grants NGS-0326386, ACI- 0234273 and NSF Career Award IIS-0347662. The authors would like to thank Dr David Thompson and Monika Jankun-Kelly from Department of Aerospace Engineering, Mississippi State University, Dr Ming Jiang from Livermore National Labs and Yootai Kim, Department of Computer Science and Engineering, Ohio State University for providing Computational Fluid Dynamics datasets and helping in analyzing the results. We also thank Dr John Wilkins, Department of Physics, Ohio State University for providing datasets for Computational Molecular Dynamics.

9. REFERENCES

[1] R. K. Adair. *The Physics of Baseball* . HarperCollins Publsiher, 2002.

[2] C. C. Aggarwal and D. Agrawal. On nearest neighbor indexing of nonlinear trajectories. In *Principles of database systems*, 2003.

[3] R. Agrawal, K.-I. Lin, H. S. Sawhney, and K. Shim. Fast Similarity Search in the Presence of Noise, Scaling, and Translation in Time-Series Databases. In *Very Large Data Bases (VLDB) Conference*, 1995.

[4] P. Bakalov, M. Hadjieleftheriou, E. Keogh, and V. J. Tsotras. Efficient trajectory joins using symbolic representations. In *International conference on Mobile data management*, 2005.

[5] Y. Cai and R. T. Ng. Indexing spatio-temporal trajectories with chebyshev polynomials. In *SIGMOD Conference*, 2004.

[6] L. Chen, M. T. Özsu, and V. Oria. Robust and Fast Similarity Search for Moving Object Trajectories. In *SIGMOD Conference*, 2005.

[7] M. J. Egenhofer. Reasoning about binary topological relations. In *Symposium on Large Spatial Databases*, 1991.

[8] D. Eggert, A. Lorusso, and R. Fisher. Estimating 3-d rigid body transformation: A comparison of four major algorithms. *MVA*, 9:272–290, 1997.

[9] C. Faloutsos, M. Ranganathan, and Y. Manolopoulos. Fast Subsequence Matching in Time-Series Databases. In *SIGMOD Conference*, 1994.

[10] M. Hadjieleftheriou, G. Kollios, V. J. Tsotras, and D. Gunopulos. Efficient indexing of spatiotemporal objects. In *Extending DataBase Technology*, 2002.

[11] M. Jiang, R. Machiraju, and D. Thompson. Geometric verification of swirling features in flow fields. In *Proceedings of IEEE conference on Visualization*, 2002.

[12] E. J. Keogh and M. J. Pazzani. A simple dimensionality reduction technique for fast similarity search in large time series databases. In *PAKDD*, 2000.

[13] Y. Kim and R. Machiraju. Swirling Images. In *OSU-CISRC-1/06- TR03*, 2006.

[14] G. Kollios, D. Gunopulos, and V. J. Tsotras. On indexing mobile objects. In *Principles of database systems*, 1999.

[15] G. Kollios, V. J. Tsotras, D. Gunopulos, A. Delis, and M. Hadjieleftheriou. Indexing animated objects using spatiotemporal access methods. *IEEE Transactions Knowledge and Data Engineering*, 13(5), 2001.

[16] F. Korn, H. V. Jagadish, and C. Faloutsos. Efficiently supporting ad hoc queries in large datasets of time sequences. In *SIGMOD Conference* , 1997.

[17] S. Mehta, K. Hazzard, R. Machiraju, S. Parthasarathy, and J. Wilkins. Detection and visualization of anomalous structures in molecular dynamics simulation data. In *IEEE Visualization*, 2004.

[18] M. F. Mokbel, T. M. Ghanem, and W. G. Aref. Spatio-temporal Access Methods. *IEEE Data Engineering Bulletin*, 2003.

[19] J. Ni and C. V. Ravishankar. Pa-tree: A parametric indexing scheme for spatio-temporal trajectories. In *International Symposium on Spatial and Temporal Databases*, 2005.

[20] J. Nocedal and S. Wright. *Numerical Optimization* . Springer, New York, 1999.

[21] D. Papadias, Y. Theodoridis, T. K. Sellis, and M. J. Egenhofer. Topological relations in the world of minimum bounding rectangles: A study with r-trees. In *SIGMOD Conference*, 1995.

[22] R. Parent. *Computer Animation- Algorithms and Techniques* . Morgan Kauffman, 2002.

[23] D. Pfoser, C. S. Jensen, and Y. Theodoridis. Novel approaches in query processing for moving object trajectories. In *VLDB*, 2000.

[24] K. pong Chan and A. W.-C. Fu. Efficient Time Series Matching by Wavelets. In *ICDE*, 1999.

[25] K. Porkaew, I. Lazaridis, and S. Mehrotra. Querying mobile objects in spatio-temporal databases. In *International Symposium on Spatial and Temporal Databases*, 2001.

[26] C. Rao and S. S. and. Statistical analysis of shape of objects based on landmark data. In *Proc Natl Acad Sci USA, 93(22):1213212136,*, 1996.

[27] S. Saltenis, C. S. Jensen, S. T. Leutenegger, and M. A. Lopez. Indexing the positions of continuously moving objects. *SIGMOD Rec.*, 29(2), 2000.

[28] D. Silver and X. Wang. Tracking and visualizing turbulent 3d features. *IEEE Trans. Vis. Comput. Graph.*, 3(2), 1997.

[29] Y. Tao, C. Faloutsos, D. Papadias, and B. Liu. Prediction and indexing of moving objects with unknown motion patterns. In *SIGMOD Conference*, 2004.

[30] Y. Tao, D. Papadias, and J. Sun. The tpr*-tree: An optimized spatio-temporal access method for predictive queries. In *VLDB*, 2003.

[31] S. Umeyama. Least-squares estimation of transformation parameters between two point patterns. *IEEE Trans. Pattern Anal. Mach. Intell.*, 13(4), 1991.

[32] H. Yang, S. Parthasarathy, and S. Mehta. A generalized framework for mining spatio-temporal patterns in scientific data. In *KDD*, 2005.

[33] H. Yang, S. Parthasarathy, and S. Mehta. Towards association-based spatio-temporal reasoning. In *IJCAI: Workshop on Spatio-temporal Reasoning*, 2005.