5-10-2003

Feature Tracking in Two Dimensional Time Varying Datasets

Sajjit Thampy

FEATURE TRACKING IN TWO-DIMENSIONAL TIME-VARYING DATA SETS

By

Sajjit Thampy

A Thesis
Submitted to the Faculty of
Mississippi State University
in Partial Fulfillment of the Requirements
for the Degree of Master of Science
in Computational Engineering
in the College of Engineering

Mississippi State, Mississippi

May 2003

FEATURE TRACKING IN TWO-DIMENSIONAL TIME-VARYING DATA SETS

By

Sajjit Thampy

Approved:

David Thompson Associate Professor of Aerospace Engineering (Major Professor)	James E. Fowler Associate Professor of Electrical Engineering (Committee Member)
Ioana Banicescu Associate Professor of Computer Science (Committee Member)	Boyd Gatlin Associate Professor of Aerospace Engineering (Graduate Coordinator)

A. Wayne Bennett
Dean of the College of Engineering

Name: Sajjit Thampy

Date of Degree: May 10, 2003

Institution: Mississippi State University

Major Field: Computational Engineering

Major Professor: Dr David Thompson

Title of Study: FEATURE TRACKING IN TWO-DIMENSIONAL TIME-VARYING DATA SETS

Pages in Study: 67

Candidate for Degree of Master of Science

This study investigates methods that can be used for tracking features in computational-fluid-dynamics datasets. The two approaches of overlap based feature tracking and attribute-based feature tracking are studied. Overlap based techniques use the actual degree of overlap between sucessive time steps to conclude a match. Attribute-based techniques use characteristics native to the feature being studied, like size, orientation, speed etc, to conclude a match between candidate features. Due to limitations on the number of time steps that can be held in a computer's memory, it may be possible to load only a time-subsampled data set. This might result in a decrease in the overlap obtained, and hence a subsequent decrease in the confidence of the match.

This study looks into using specific attributes of features, like rotational velocity, linear velocity to predict the presence of that feature in a future time step. The use of predictive techniques is tested on swirling features, i.e., vortices.

An ellipse-like representation is assumed to be a good approximation of any such feature. The location of a feature in previous time-steps are used to predict its position in a future time-step. The ellipse-like representation of the feature is translated over to the predicted location and aligned in the predicted orientation. An overlap test is then done. Use of predictive techniques will help increase the overlap, and subsequently the confidence in the match obtained. The techniques were tested on an artificial data set for linear velocity and rotation and on a real data set of simulation of flow past a cylinder. Regions of swirling flow, detected by computing the swirl parameter, were taken as features for study. The degree of overlap obtained by a basic overlap and by the use of predictive methods were tabulated. The results show that the use of predictive techniques improved the overlap.

ACKNOWLEDGMENTS

I would like to thank my major professor, Dr. David Thompson for his patience, guidance and extensive support that he meted out to me for helping me in my work. I would also like to thank Dr. Raghu Machiraju, Dr. James Fowler, committee member Dr. Ioana Banicescu and other members of the EVITA team who helped me gain more insight into the problem at hand. In addition I would like to thank all members/staff of the ERC, who helped create an environment conducive for work. This work was supported by the NSF Large Data and System Software Visualization Program (9982344).

TABLE OF CONTENTS

Page

ACKNOWLEDGMENT ii

LIST OF TABLES v

LIST OF FIGURES vi

NOMENCLATURE viii

CHAPTER

I. INTRODUCTION 1
1.1 The EVITA project 1
1.2 Feature Tracking 5
1.3 Objective of Work 6
1.4 Overview of Work 6

II. DETECTING THE FEATURES 8
2.1 Calculation of the Swirl Parameter 9
2.2 Generation of Feature Map 12
2.3 Feature Segmentation 13
2.3.1 The scan pass algorithm 14
2.3.2 Feature Representation 18

III. FEATURE TRACKING 19
3.1 Feature States 19
3.2 Silver's Approach to Feature Tracking 20
3.2.1 The Brute-Force Implementation 21
3.2.2 Quad-tree-Based Implementation 22
3.2.3 Silver's Tracking Algorithm Implementation 25
3.3 Attribute-Based Feature Tracking 30

CHAPTER Page

3.3.1 The Sense of Rotation 31
3.3.2 Evaluating Correspondence 33

IV. PREDICTIVE FEATURE TRACKING 35
4.1 Predictive Techniques 35
4.2 Motion of Features 36
4.3 The Issue of Various Grid Types 37
4.4 The General Approach 40
4.5 Translation 41
4.6 Rotation 43

V. RESULTS AND DISCUSSION 45
5.1 Artificial Data Sets 45
5.2 Flow Past Cylinder 58

VI. CONCLUSIONS AND FUTURE WORK 64
6.1 Future Work 65

REFERENCES 66

LIST OF TABLES

TABLE Page

5.1 R values for feature ID 0,undergoing uniform acceleration, using basic overlap and prediction using information from the previous two and three time steps 46

5.2 R values for feature ID 0,undergoing uniform rotation, using basic overlap and prediction using information from the previous two and three time steps 52

LIST OF FIGURES

FIGURE Page

1.1 Schematic Representation of the EVITA system [1] 4

2.1 Illustration of Individual Features 13

3.1 Example of a case when overlap does not imply a match. 21

3.2 Illustration of the Brute-Force Technique. 22

3.3 Illustration of the Quad-tree Data Structure 23

3.4 The Brute-Force Technique. 24

3.5 The Quad-tree Technique. 24

3.6 Initialized Overlap Table. 25

3.7 Generation of Overlap Table. 26

3.8 A Problematic Case $t_1 = t$, $t_2 = t + \Delta t$. 27

3.9 Using the sense of rotation as an attribute. 32

3.10 Strength Based Comparison. 32

4.1 The Predictive Scheme 37

4.2 The Ellipse Representation 38

5.1 Snapshot of the Artificial Dataset using predictive tracking (time step = 1) 47

5.2 Snapshot of the Artificial Dataset using predictive tracking (time step = 2) 48

5.3 Snapshot of the Artificial Dataset using predictive tracking (time step = 3) 49
5.4 Snapshot of the Artificial Dataset using predictive tracking (time step = 4) 50
5.5 Snapshot of the Artificial Dataset using predictive tracking(time step = 5) 51
5.6 Snapshot of overlap with rotation prediction (time step = 2) . . . 53
5.7 Snapshot of overlap with rotation prediction (time step = 3) . . . 54
5.8 Snapshot of overlap with rotation prediction (time step = 4) . . . 55
5.9 Snapshot of overlap with rotation prediction (time step = 5) . . . 56
5.10 Snapshot of overlap with rotation prediction (time step = 6) . . . 57
5.11 Snapshot of the Grid used for flow past a cylinder 58
5.12 Plot of R value for $\Delta t = 1...10$ for the 3^{rd} time step using 2^{nd} and 1^{st} 59
5.13 Plot of R value for $\Delta t = 1...10$ for the 4^{th} time step using 3^{rd}, 2^{nd} and 1^{st}. 60
5.14 Snapshot of the use of predictive techniques for time $t = 37$ using features from $t = 31$ and $t = 25$ 61
5.15 Snapshot of the use of predictive techniques for time $t = 43$ using features from $t = 37$ and $t = 31$ 62
5.16 Snapshot of the use of predictive techniques for time $t = 43$ using features from $t = 37$, $t = 31$ and $t = 25$ 63

NOMENCLATURE

Identifiers:

$\tilde{B}$	Embedded Bitstream
$\tilde{S}$	Significance Map
L	Velocity gradient vector
Λ	Diagonal matrix of the eigen vector of L
τ	Swirl value
s_{ij}	Feature map
F_t	Feature forest
t	Time step
Δt	Increment in time step
C_{func}	Correspondence function
T_i	Tolerance for the correspondence function
W_i	Weight for C_{func}
N_{func}	Number of contributing functions
O_x	A feature representation
(x_p, y_p)	The predicted centroid of a feature

CHAPTER I

INTRODUCTION

Computer programs that are used to simulate temporally varying physical phenomena typically produce large volumes of data. In computational field-simulation applications, the disparate spatial and temporal scales required to simulate complex physical phenomenon dictate the growth of datasets. For example, if there are five floating-point values to be computed for every node within a 100x100x100 grid, there are $5x10^6$ floating-point numbers that must be stored for a single solution. Unfortunately, the development of data-management and visualization techniques has not kept pace with the size and complexity of these generated datasets. The end users of such systems are usually overwhelmed with large volumes of computed data. Discerning interesting patterns in them could turn out to be a difficult task. The problem is more challenging when the end users have to deal with temporally varying data that require storage of multiple time steps.

1.1 The EVITA project

The EVITA (Efficient Visualization and Interrogation for Terra-scale Datasets) project [1] attempts to address some of the above issues. The goal of the project is to develop a prototype visualization system based on a representational scheme which facilitates ranked access to macroscopic features in compressed

representations within large datasets. By doing so, it gives the end user a means to interactively transform data into pictorial forms in a manner which fosters discovery.

For effective tera-scale visualization, data-compression is unavoidable. While choosing the data-compression technique, care must be taken to ensure that the compressed representation is amenable to computing parameters like gradients and vorticity in the compressed domain. Secondly, the scheme needs to be lossless and hence allow for a complete reconstruction of the domain from the compressed representation. Finally, visualization is an issue as image-rendering time comes at a price. The scheme should allow for rendering phenomena from a partial reconstruction of the compressed dataset. Put in other words, the degree of inflation should be open to the user depending on the system used to execute the code.

The EVITA project focuses on features of interest in computational-fluid-dynamics datasets. The system is built on a client-server model with three main components: the pre-processor, the server, and the client. In the pre-processor the data and grid are compressed using wavelet techniques while a feature-detection algorithm is used to contextually rank and identify features directly in the wavelet domain. The compressed representation is then transmitted to the client side. This saves memory on the client side as it only needs to process the ranked features that are transmitted. The schematic layout is shown in figure 1.1.

The pre-processor takes as input a structured rectilinear or curvilinear grid and its associated field values. The *Feature Detector* module performs an automatic

feature detection and stores a scalar value for each point on the grid. This value indicates whether that particular node is significant and its degree of significance. The *Significance Map Encoder* takes the scalar values at each node and generates the significance map. The *Significance Map Generator* identifies regions of interest (ROI) and labels and ranks them based on predefined criteria. The *Wavelet Transform* module generates multi-scale wavelet coefficients for the field. The *Embedded Encoder* takes in the grid, the wavelet transformations, and the significance map as input and generates an embedded bitstream $\tilde{B}$. The embedded bitstream consists of bitplanes of compressed ROI's in a particular order. The compression system chosen for encoding should be suitable for both scalar and vector fields and must be lossless allowing perfect reconstruction at the client side. However, depending on user input, the compressed representation should also allow for partial reconstruction at the user end. All the computational work is done in a offline manner.

When the client initiates a request, the server transmits the bitstream $\tilde{B}$ to the client. In order to transmit the bitstream in a manner requested by the client, a *Transcoder* is built in. A *Transcoder* is defined as a processor which converts one form of coding to another. Information regarding the ranking of the features is obtained by decoding the significance map $\tilde{S}$ by the *Significance Map Decoder*. When the visualization is initiated, the server transmits each bitplane in the same order as received in $\tilde{B}$. The client has the ability to change the rankings of the features by sending requests to the *Priority Scheduler* through the *Controller*. The *Transcoder* relocates the bitplanes in $\tilde{B}$ based on the requested priority. The client side decodes the bitplanes transmitted by the server using the significance map and the priority schedule. The *Embedded Decoder* performs the

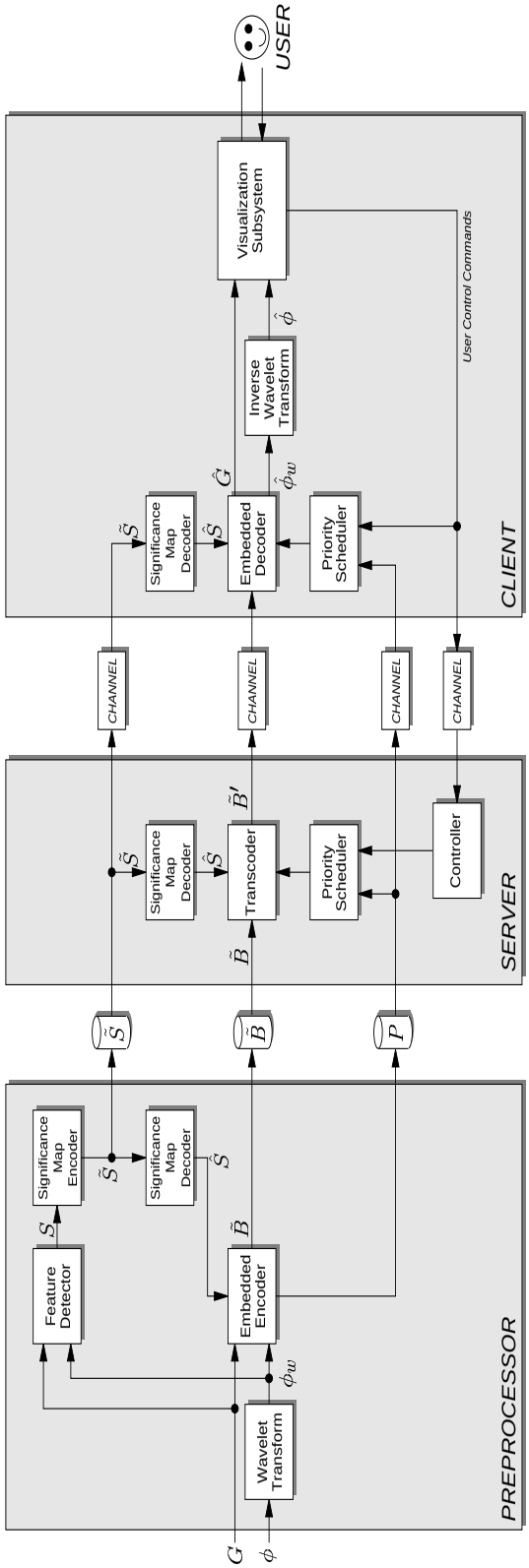

Figure 1.1: Schematic Representation of the EVITA system [1]

inverse operation of the *Transcoder* and the *Embedded Decoder* on $\tilde{B}$. It produces a wavelet transformation on the wavelet coefficients to generate the ROI's. This data is transfered to the *Visualization Subsystem*. The *Visualization Subsystem* is responsible for rendering the features. The *Priority Scheduler* in the client is responsible for communicating the user control commands to the server.

Visualizing several time-varying features on a single screen may be difficult to comprehend. Therefore, a good functionality to have in the system is the capability to automatically track a selected ROI across multiple time steps. This will help the user better understand the phenomena being visualized. This necessitates the development of tracking techniques for the EVITA system.

1.2 Feature Tracking

One of the key functions of the EVITA system is tracking features in a series of two-dimensional time-varying data sets. Tracking a series of two-dimensional images has been addressed to a great extent in computer vision for motion estimation [2]. Typically, the tracking process involves a matching process between two or more images using pixels, points, lines, and blobs, based on their motion, shape, and visual information. Aggarwal [3] uses correlation-template-tracking techniques. A subset frame of the first frame is detected. The regions are taken to be rectangular for convenience. The template is then used to correlate objects in subsequent frames. For non-rigid objects, the template is adjusted to represent regions that have changed over time. The feature-tracking begins with feature extraction and then correlation of those features with other features in subsequent time steps. The feature tracking problem is very much like the *Correspondence*

Problem. Silver *et al.* [4] describe a generalized tracking algorithm that can be used on unstructured two- and three-dimensional meshes. This technique uses the principle of area/volume overlap to determine a best match between features. A more detailed explanation of this technique will be given in chapter III. Reindeers *et al.* [5] describe a method for tracking features based on feature attributes such as velocity, rotation, and other characteristics that can be extracted from the phenomena under study. A detailed explanation is given in a Chapter III.

1.3 Objective of Work

The objective of the present effort is to enhance existing feature-tracking techniques. An approach to predict the physical characteristics of a feature in a time step based on its immediate past history is studied. Several techniques have been discussed in [4] [5] for tracking features in time. The basic assumption in both is that features evolve consistently and that the data sets are well sampled in time. However, this need not always be the case. In many instances, only every fifth or tenth time step of a simulation may be saved. It is also possible that the features may not evolve consistently and may undergo irregular forms of rotation, translation, etc.

1.4 Overview of Work

Chapter II deals with an overview of techniques for detecting features and segmenting them in a manner that is amenable to representation in a data structure. In many computational-fluid-dynamics data sets, the vortex is an important feature to capture. For the purpose of this study, the swirl parameter [6] is used to detect vortices. The swirl parameter gives an estimate of the

tendency for flow at the given node to undergo a swirling motion. It is computed locally for every node. Various segmentation techniques, such as scan pass and region growing, are then discussed for storing the features in useful data structures.

Chapter III discusses in detail the existing methods for feature tracking. The various states a feature can be in are discussed. Attribute-based feature tracking and the generalized tracking algorithm are discussed. Chapter IV discusses the use of predictive techniques in tracking rapidly evolving or translating features. Chapter V enlists the results obtained by testing the technique on artificial data sets and real data sets. The real data set is a simulation of a flow past a cylinder. Conclusions and suggestions for future work is presented in Chapter VI.

CHAPTER II

DETECTING THE FEATURES

A feature within a data set defines a region of interest. There are many ways to define a feature. In the case of a digital image, all pixel values that fall within a prescribed gray-scale range could be classified as important, and the rest can be written off as not significant. For a scientist studying bathymetric data, depths within a range of values could be of relevance for the study. In other words, we need to develop a consensus as to how we should define a feature from the given data field, so that it is relevant for understanding the physical phenomena that we are trying to study.

In many computational-fluid-dynamics simulations, vortices are significant features. For the purpose of this study, the swirl parameter [6], is computed for a velocity-vector grid. This parameter gives an estimate on the amount of local rotating motion that is present in a given velocity field and helps to locate vortices in the flow. After the swirl parameter is computed at all nodes in the grid, a feature map is generated by marking nodes that have swirl values larger than a prescribed threshold. This is followed by the process of segmentation, which involves segmenting features in the domain and tagging them with ID's for future access. There are many methods in literature for segmenting features [7] [8]. The *scan-pass* algorithm and the *region growing* algorithm are discussed in this chapter.

2.1 Calculation of the Swirl Parameter

Swirling flow is defined as any fluid motion that rotates about a common center, not necessarily fixed. Berdahl and Thompson [6] define a derived scalar parameter called the *swirl*. As this parameter estimates the strength of rotating motion, it can be used on a velocity vector field to detect vortices and eddies.

The eigenvalues of the velocity gradient tensor are complex in regions of swirling flow. The velocity gradient tensor fully describes the spatial variation of velocity vector to the first order. Thus, the variation of the velocity of a particle at a position x with respect to the origin is

$$\frac{dx}{dt} = Lx \tag{2.1}$$

where L is the velocity gradient vector defined as

$$L = \begin{bmatrix} \frac{\partial u}{\partial x} & \frac{\partial u}{\partial y} & \frac{\partial u}{\partial z} \\ \frac{\partial v}{\partial x} & \frac{\partial v}{\partial y} & \frac{\partial v}{\partial z} \\ \frac{\partial w}{\partial x} & \frac{\partial w}{\partial y} & \frac{\partial w}{\partial z} \end{bmatrix}.$$

If L has a complete set of linearly independent eigenvectors, a similarity transform exists such that

$$L = T\Lambda T^{-1}$$

where Λ is the diagonal matrix of the eigenvalues of L and is defined as

$$\Lambda = diag(\lambda_1, \lambda_2, \lambda_3).$$

The matrix T is

$$T = \begin{bmatrix} r_1, & r_2, & r_3 \end{bmatrix}$$

where (r_1, r_2, r_3) are the column eigenvectors of L, and the matrix T^{-1} is given by

$$T^{-1} = \begin{bmatrix} l_1 \\ l_2 \\ l_3 \end{bmatrix}$$

where (l_1, l_2, l_3) are the row eigenvectors of L. Assuming that the elements of L are spatially and temporally constant and by substituting $X = T^{-1}x$ in equation 2.1 gives

$$\frac{dX}{dt} = \Lambda X.$$

The system is now decoupled since Λ is a diagonal matrix, and the solution is

$$X(t) = \begin{bmatrix} X_1(t) \\ X_2(t) \\ X_3(t) \end{bmatrix}$$

$$= \begin{bmatrix} e^{\lambda_1 t} X_{01} \\ e^{\lambda_2 t} X_{02} \\ e^{\lambda_3 t} X_{03} \end{bmatrix}. \tag{2.2}$$

The solution given in equation 2.2 consists of three decoupled motions in the complex plane. The quantities X_{01}, X_{02}, and X_{03} represent the initial points of the three trajectories. If λ_1 and λ_2 are the complex-conjugate pair, and if λ_3 is real as a consequence, then $X_1(t)$ and $X_2(t)$ are spiraling motions in the complex plane with amplitudes that decay or grow depending on the sign of the real part of the complex-conjugate eigenvalues. If the real part is zero, circular paths in the complex plane are obtained. A period for this motion can be computed as

$$t_{orbit} = \frac{2\pi}{|Im(\lambda_{1,2}|)}. \tag{2.3}$$

The remaining eigenvalue λ_3 yields an exponentially decaying or growing solution $X_3(t)$ along the real axis. After performing an inverse transform, the solution can be written as

$$x(t) = r_1 e^{\lambda_1 t} X_{01} + r_2 e^{\lambda_2 t} X_{02} + r_3 e^{\lambda_3 t} X_{03}.$$

The right eigenvectors map the three motions in the complex plane to the motion in the physical plane. For a two-dimensional case, the real eigenvalue is zero, and the third dimension decouples because of the forms of the left and right eigenvectors. The existence of complex eigenvalues of the velocity gradient tensor in a region is a necessary but not sufficient condition for the presence of swirling motion. The intrinsic swirl parameter τ is defined as the ratio of the time it takes a particle to convect through a region of complex eigenvalues to the orbit time

$$\tau = \frac{t_{conv}}{t_{orbit}}. \tag{2.4}$$

If l is the characteristic length associated with the size of the region of complex eigenvalues, and V_{conv} is the magnitude of the velocity aligned along l then

$$t_{conv} = \frac{l}{V_{conv}}. \tag{2.5}$$

Substituting equation 2.4 and 2.3 in 2.5 gives

$$\tau = \frac{|Im(\lambda_{1,2})|l}{2\pi V_{conv}}. \tag{2.6}$$

This result indicates that for small values of τ, the fluid convects too rapidly through the region of complex eigenvalues to be captured in swirling motion. In regions of large τ, the fluid is trapped in a swirling motion. In two dimensions, the convective velocity lies in the plane of motion. Also the characteristic length is normalized to unity. The *swirl* parameter τ is computed locally for every node in the data set. More details can be found in [6].

2.2 Generation of Feature Map

The feature map s_{ij} is generated by tagging nodes on the grid that have swirl values above a specified threshold τ_{min}. It can be represented as:

$$s_{ij}^t = \begin{cases} 1 & \text{if } \tau \geq \tau_{min} \\ 0 & \text{otherwise} \end{cases}$$

where t is the time-step in the dataset that the swirl significance map represents. It may be noted that some phenomena may not be adequately sampled. It is possible

that some nodes are marked significant for only a few time steps. An appropriate noise filter would be needed to filter out such nodes.

2.3 Feature Segmentation

The previous section describes the generation of the swirl significance map. However for the purpose of analysis, the swirl significance map is of little use, as it does not exist in a format to give structure to the physics of the phenomena that the user is investigating. In order to better discern the evolving phenomena, we need to identify the significant nodes that are physically contiguous. This is shown in the figure 2.1. However, using this idea of contiguity, it is not possible to

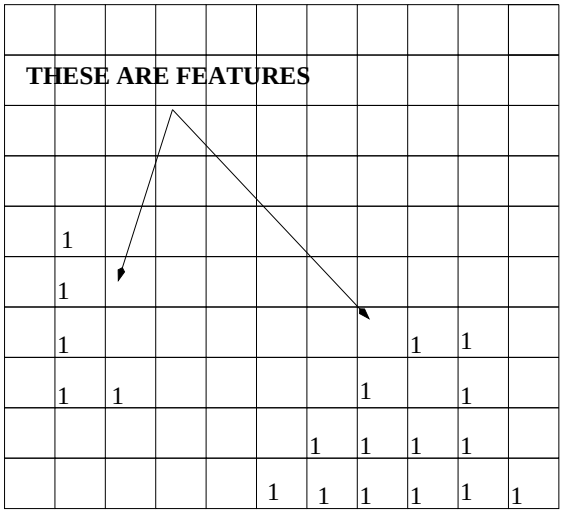

Figure 2.1: Illustration of Individual Features

correctly segment two distinct features that are touching each other. Alternative techniques would have to be used to discern whether the touching features are in reality one feature or two. This problem is beyond the scope of this effort.

This leads us to the definition of a feature as a set of nodes that are defined to be significant and are contiguous. Assume f_t is a feature which is initially a null set, where t is the time step, i.e.,

$$f_t = \varnothing,$$

if s_{ij} is significant, then

$$f_t \leftarrow f_t \cup \{s_{ij}\}$$

and for every significant s_{kl} in the logical neighborhood of s_{ij}, the following is true

$$f_t \leftarrow f_t \cup \{s_{kl}\}.$$

When applied recursively over the entire domain, we get a set of f_t which is called the *Feature Set* F_t defined over the entire domain for that particular time-step t. Stated formally,

$$F_t = \{f_t\}.$$

There are several ways to implement the aforementioned steps including *region growing* and the more traditional *scan pass* algorithm. The *scan pass* algorithm is not directly intuitive, however the *region growing* algorithm is very much intuitive.

2.3.1 The scan pass algorithm

This subsection describes the *scan pass* algorithm [7] [8]. It takes as input the swirl significance map and outputs a segmented swirl significance map. The scan-pass algorithm works for structured curvilinear meshes. Every node is visited, and is assigned a tag ID. The tag ID is allocated based on the tag ID's of the node's

neighbors. A node is given the same ID as its neighbor previously visited. If there are no neighbors with a tag ID, a new ID is allocated. The next stage is a recursive merge where in all the connected tag ID's are assigned to the lowest ID in the group. The end result is that all connected nodes have the same tag ID.

```
// The scan pass algorithm
// First pass assigns labels to each point:NxN is the grid size
 id = 1
 for j = 0 to N
  for i = 0 to N
   if(s_(i,j)!= NO_SWIRL)
   {
    possible_ids = number_of_non_zero_neighbor_ids();
    if ( possible_ids > 1 )
    {
     mask(i,j) = Minimum(mask(i-1,j-1),mask(i+1,j-1),mask(i,j-1),mask(i-1,j));
     for each neighbor_id in possible_ids
      newid(neighbor_id) = mask(i,j);
    }
    else
    {
     mask(i,j) = id++;
    }
  end i;
 end j;
```

```
// Merge newid labels using recursive calls.

 for i = 0 to id
  newid(i) = recursive_merge(newid(i));
 end i;

// Now replace old labels with new ranked labels.
 for j = 0 to N
  for i = 0 to N
   mask(i,j) = newid(mask(i,j));
  end i;
 end j;
```

The *region growing* algorithm [4] is more intuitive than the *scan pass* algorithm. A recursive technique is used in this algorithm. All significant nodes are marked unvisited at the beginning. A *seed* is chosen as one of the nodes that has been defined as significant. This choice can be made interactively or by the system. All the unvisited neighboring nodes of this seed are added to the feature and marked as visited. The algorithm recurses with all the neighbors as a *seed*. The recursion continues until there are no more neighbors to grow into.

```
// The region growing algorithm.
// node_list is a list of nodes, feature_list is a list of features.
 for i = 0 to N
  for j = 0 to N
   if ( s(i,j) != NO_SWIRL)
```

```
    mark s(i,j) as visited;
    grow(i,j);
  end i;
 end j;
// the grow() routine is mentioned here.
grow(i,j)
{
 add s(i,j) to node_list;
 for each significant logical neighbour of (i,j) not visited.
  mark (i,j) as visited;
  grow(i,j);
 end;
}
```

It may be noted that the *scan pass* algorithm runs through the entire set of nodes twice whereas the *region growing* runs through them once. This makes the execution of the *region growing* algorithm somewhat faster than the *scan pass* algorithm. However, the *region growing* algorithm, being a recursive function, could have the problem of a function stack memory overflow as the depth of the recursion may become large depending on the size of the feature the algorithm is trying to segment. However this can be overcome by placing a tab on the depth of recursion depending on the available system memory, followed by a recursive merge of adjacent features logically connected together. As this method uses recursive merge to a small extent, it is in some sense, is a hybrid of both the algorithms mentioned above.

2.3.2 Feature Representation

Segmenting the nodes into individual features is the first step to study evolving phenomena. This is necessary, but not sufficient. This signals a need for them to be organized in a manner that is general enough to handle a wide variety of shapes and features. Simplicity of representation also needs to be factored in, as it would ease tracking and quantification. Physically based models have been used for tracking in computer vision [2]. Work was also directed for representing features by their centroids and second-order moments [7].

To be of any use for analysis, the segmented features need to be associated with information regarding their position (for example the centroid (x_{cen}, y_{cen})) and the time-step they belong to t. An ellipse-like representation of the feature is also stored. This includes, the lengths of the major and minor axis and the orientation of the axes. The feature is also given a tag integer ID α, so that they are easily available for look up by other routines.

CHAPTER III

FEATURE TRACKING

Chapter II discusses the issues related to defining features on datasets and procedures to segment and classify features in a meaningful manner to make them amenable for tracking. In this chapter, we shall discuss the various methods that have been used to track these segmented features across multiple time-steps. Tracking relevant features is a useful visualization tool for studying evolving phenomena as it gives greater insight into the physics of the problem.

3.1 Feature States

Having defined a feature, we can classify any evolving feature in a time-varying dataset as either undergoing *Creation, Continuation, Bifurcation, Amalgamation* or *Dissipation* over time [7]. *Creation* occurs when a feature appears in a data set at time-step t and cannot be matched to a feature in data set at the previous time-step $t - \Delta t$. *Continuation* occurs when a given feature continues to exist, not necessarily in the same location, shape and size, in a time-step t from the previous time step $t - \Delta t$. *Bifurcation* is the state when a particular feature in time step t breaks up into two or more features in time step $t + \Delta t$. *Amalgamation* is the state when two or more features from time step t merge into one feature in time step $t + \Delta t$. *Dissipation* is the state when a feature in time step t ceases to exist in time step $t + \Delta t$. It may be noted that *Amalgamation* in forward time is *Bifurcation*

in backward time. Similarly *Creation* in forward time is *Dissipation* in backward time. Silver [7] notes that it is sufficient to capture *Continuation, Bifurcation*, and *Creation* in forward time, *Amalgamation* and *Dissipation* can be captured by running the same algorithm back-wards.

3.2 Silver's Approach to Feature Tracking

Matching a feature in one time-step to the next is called the correspondence problem. Silver *et al.* [4] [7] use a simple idea that a match among features in adjacent time-steps can be generated using an area-overlap test. A matching metric can be defined as some parameter that gives a quantitative measure on the degree of correspondence. Typically, the metric would need to be maximized. Many simple techniques have been proposed for feature tracking using elementary matching metrics. If there are two features O_a and O_b, the degree of area overlap can be computed as follows

$$R = \frac{Area(O_a \bigcap O_b)}{\sqrt{Area(O_a)Area(O_b)}}. \tag{3.1}$$

In practice the degree of area overlap needed to imply a match is taken to be greater than a specified tolerance, i.e.,

$$R > Tolerance.$$

This condition can be thought to be necessary but it is not sufficient. There may be cases where an overlap need not imply a match.

An example of such a case is shown in figure 3.1. A set of features may translate uniformly causing the overlap criterion to produce erroneous results.

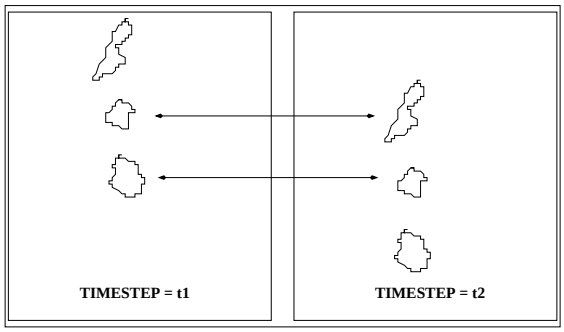

Figure 3.1: Example of a case when overlap does not imply a match.

3.2.1 The Brute-Force Implementation

The simplest form of a tracking algorithm described in [4] is basically a brute-force technique. In this technique, the bounding box of every feature in time step t is compared with the bounding box of every other feature in time step $t + \Delta t$. A node by node comparison is made if and only if the bounding boxes of the features overlap. This saves some time. Yet another assumption that is made for this technique is that the datasets are closely sampled over time, and so the features do not change or move to a large extent to make overlap impossible. In summary, the feature-tracking algorithm works as a two-stage process.

1. Determine the set of features from adjacent time-steps that overlap.

2. Perform a best-matching test on the features that overlap. An area-overlap test can be used to determine the best match.

Figure 3.2 illustrates this technique. The comparisons made are $(A-1, A-2, A-3, A-4, A-5)$ and $(B-1, B-2, B-3, B-4, B-5)$. The number of comparisons

made grows exponentially with the number of features n. The complexity becomes $O(n^2)$ and hence the method is inefficient.

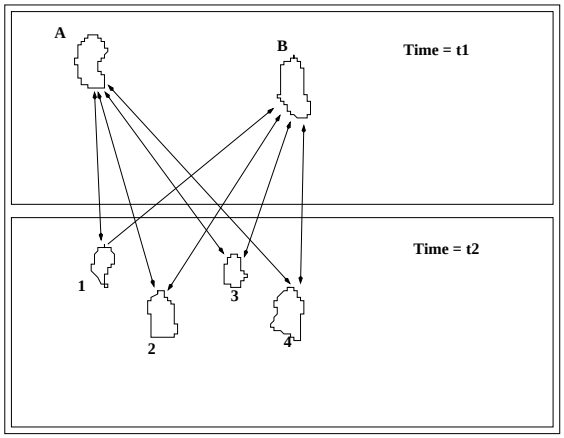

Figure 3.2: Illustration of the Brute-Force Technique.

3.2.2 Quad-tree-Based Implementation

If an area-overlap test is chosen, a quad-tree-based data structure is a good choice for spatially partitioning the features present in a time step. The features are given unique feature ID's and stored in an quad-tree. The union of all the quad-trees representing the features in an entire dataset is called a *quad-tree forest*. The quad-tree data structure is useful for algorithms which require the spatial location of a given point within a two-dimensional domain. The basic principle of a quad-tree is to divide a planar region into sub-domains and then to recursively partition the sub-domains into smaller sub-domains, until each sub-domain contains a suitably uniform subset of the phenomena under surveillance as shown in figure 3.3. The quad-tree and related types of partitions have many applications in computational geometry and geometric applications, including data clustering, shape representation, molecular modeling and mesh

generation.

The quad-tree-based tracking algorithm operates on two-dimensional structured meshes. Although the basic idea is extensible, the implementation here cannot be extended to unstructured meshes. Sampling an unstructured mesh at nodes and transforming into a regular mesh is a possibility. However, there would be a loss of accuracy due to aliasing.

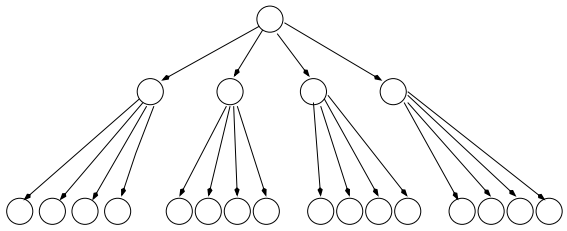

Figure 3.3: Illustration of the Quad-tree Data Structure

The number of times the sub-domain needs to be split can be based upon a predetermined knowledge of the size of the feature. The figure 3.4 shows a hypothetical case of features in consecutive time-steps. The arrows show the number of comparisons that need to be made if the brute-force technique is used. Its evident from figure 3.4 that five comparisons are needed to establish a best match. Note that the comparisons between a-1,b-2 and b-3 are not necessary because they are not located in the immediate vicinity of each other in the domain.

The quad-tree data structure stores the features based on information about the location within a domain such as the coordinates of its centroid. This is illustrated in figure 3.5. Feature a is in sub-domain I, hence it would be compared with features in the same sub-domain in the next time step. The same goes for b

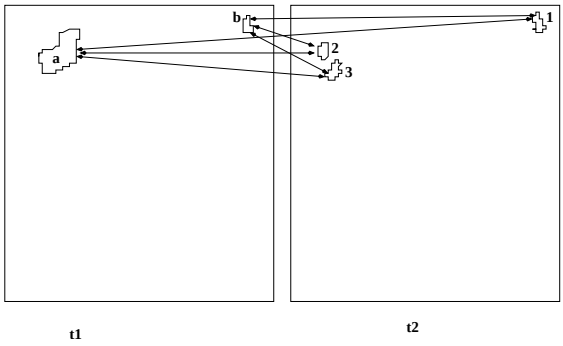

Figure 3.4: The Brute-Force Technique.

which is in II.2. It is evident from figure 3.5 that the number of comparisons have now dipped from 5 to 3.

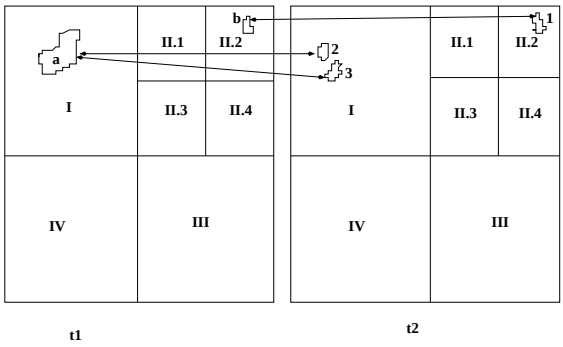

Figure 3.5: The Quad-tree Technique.

The technique, however, has its pros and cons. Consider the case when the tree is not deep enough. This would result in a congestion of features within a sub-domain. If, on the other hand, the domain was decomposed to several different sub-domains, we would not maximize the gains of using the tree-based technique. This is because a feature may end up residing in several sub-domains and time would be spent traversing the tree in order to get an estimate of the area overlap.

In order to maximize the gains from a domain-decomposition technique, the depth of subdivision needs to be ascertained before the tracking process is initiated. In order to get this number, an estimate of the average size of the feature with the

entire dataset needs to be ascertained. Only sub-domains which have features in them need be subdivided. Thus, techniques for local decomposition of sub-domains also need to be used.

3.2.3 Silver's Tracking Algorithm Implementation

The algorithm described in [4] [7] uses the techniques discussed in this section. The features from time step t_i and t_{i+1} are used to compute the overlap between them. Every feature is given an integer feature ID. Let there be n_i features in time step t_i and n_{i+1} in time step t_{i+1}. An overlap table is created with n_i rows and n_{i+1} columns. The table entry values are set to zero initially. This stage is shown in the figure 3.6. The purpose of the overlap table is to evaluate which of the features in adjacent time-steps overlap, and, if so, what is the degree of overlap.

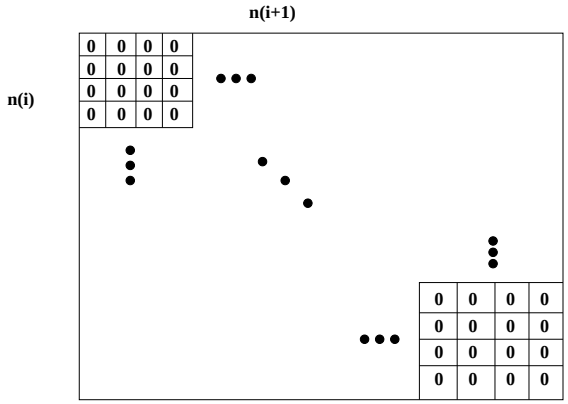

Figure 3.6: Initialized Overlap Table.

Every node within the grid stores information on whether or not that particular node is significant, and if so, the feature ID of the feature it is associated with. An assumption made here is that the grid connectivity does not change between time t_i and t_{i+1}. The nodes from both time-steps are compared in a scan-pass fashion. If a node is significant at both times the corresponding entry in the

overlap table, marking the feature ID in time step t on the rows, and the feature ID in $t + \Delta t$ denoting the column are incremented by one. The individual entries within the overlap table give us an estimate of the degree of area overlap as defined by $O_a \bigcap O_b$.

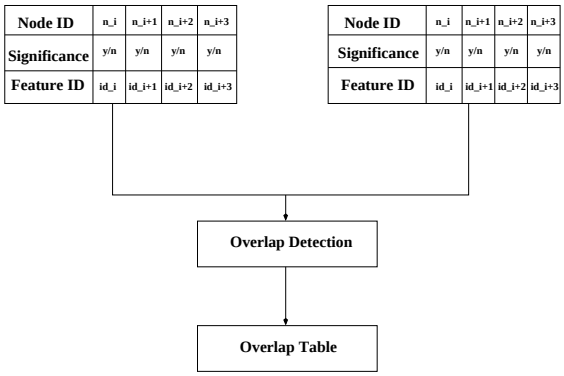

Figure 3.7: Generation of Overlap Table.

The overlap detection is implemented by having index pointers to the first node of each grid representing the time steps t and $t + \Delta t$. These pointers are incremented serially until the entire grid is traversed. The overlap detection is done before incrementing them. The algorithm is described below.

```
Overlap Detection
{
 p1 <- 0 /* set p1 to point to beginning of time step t1 */
 p2 <- 0 /* set p2 to point to beginning of time step t2 */
 while(p1<Number of Nodes and p2<Number of Nodes)
  {
    R1 = p1;R2 = p2;
    if(R1 and R2 are significant)
     {
      OverlapTable[R1.ID][R2.ID]++
```

```
      }
    p1++;p2++;
   }
}
```

The worst-case complexity for the overlap-detection algorithm is $O(p+q)$, where p is the number of significant nodes associated with features within time step t, and q is the number of significant nodes associated with features in time step $t + \Delta t$. The next stage to this process is the best-matching process. The table generated from the overlap-detection process is used for the best-matching process. The maximum-intersection criteria is used for the best-matching process. The features are considered matched if their normalized correspondence metric, equation 3.1, is maximized and also satisfiers a tolerance constraint. The tolerance value chosen depends on the nature of the datasets and the sampling frequency. A simple technique that can be used is to check for all row values within the overlap table which satisfy tolerance constraints for bifurcation and continuation, and column values for amalgamation. However this technique has some problems.

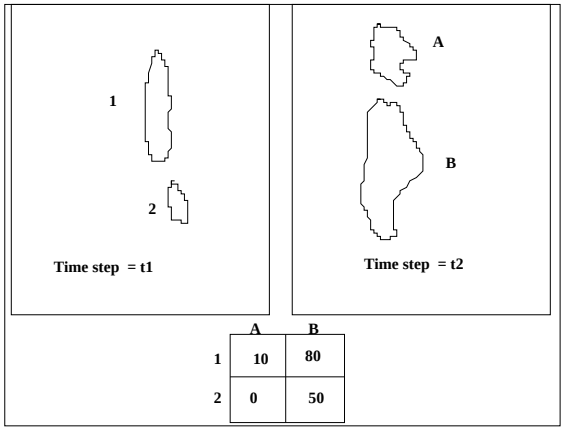

Figure 3.8: A Problematic Case $t_1 = t$, $t_2 = t + \Delta t$.

In the figure 3.8, the technique would yield the incorrect result that feature 1 splits into features A and B. A technique is needed that would process the information in the scoreboard and indicate which is a more probable event. Silver *et al.* define what they call a *scoreboard*. The scoreboard has the same size as the overlap table but stores normalized areas of intersection. The rows in the scoreboard represent features from time step t, and the columns represent objects from time step $t+\Delta t$. The next step is to generate a list of all nonzero values within every row and column. This is called the overlap list. All possible combinations of entries from the overlap list are taken. This list of all possible combinations gives a handle on what are the various possibilities of bifurcation and continuation that could have happened. This is followed by computation of the normalized area of intersection for each of the combinations. The lower scoreboard values are replaced by higher ones. This step is included so that the combination which has the highest degree of area overlap would dominate in the end. The same process is done for all the columns. Once the scoreboard is computed, we are in a position to perform feature correspondence. Entries with the same score in a row indicate bifurcation, and entries of the same score in columns indicate amalgamation. Entries which were not used in the above process are continuation. The remaining that do not satisfy tolerance constraints are tagged off as dissipation and creation in time steps t and $t + \Delta t$ respectively. The algorithm is described below.

```
Best Matching
{
 /* Initialize ScoreBoard to zero */
 For each feature in t1
  check OverlapTable for non zero entries in the row.Store it in OverlapList
```

```
   For all combinations of features within OverlapList
    Compute R the degree of area overlap.
    For each feature within OverlapList
     if R > ScoreBoard value for that entry
       Replace ScoreBoard value with R.

 For each feature in t2
  check OverlapTable for non zero entries in the colum.Store it in OverlapList
   For all combinations of features within OverlapList
    Compute R the degree of area overlap.
    For each feature within OverlapList
     if R > ScoreBoard value for that entry
       Replace ScoreBoard value with R.

 /* identify tracking events */

 Same ScoreBoard values within a row which satisfy tolerance constraints
  are classified as bifurcation
 Same ScoreBoard values within a column which satisfy tolerance constraints
  are classified as amalgamation
 Highest ScoreBoard value which is still in the search space as continuation.
 Remaining in t1 as dissipation.
 Remaining in t2 as creation.
}
```

This generalized tracking algorithm creates an event graph. The event graph maps features in time-step t to features in time-step $t + \Delta t$. The brute-force technique is not an efficient way for tracking features in time-varying datasets. The quad-tree-based technique is useful for structured meshes. However, it may not be useful if the mesh is unstructured. In unstructured meshes, there could be triangles/quadrilaterals that are present in adjacent sub-domains. Assigning which triangle/quadrilateral goes into which sub-domain will slow the process of domain decomposition. The generalized tracking algorithm by Silver *et al.* [7] is a technique that will work on unstructured meshes. However, it assumes that the features evolve consistently, and that the data sets are well sampled in time.

3.3 Attribute-Based Feature Tracking

The technique described above basically uses position as a measurable physical characteristic of the feature to compare it with features in subsequent time steps. The maximizing parameter given in equation 3.1 was the degree of area overlap. This technique does not exploit other characteristics of the physical phenomenon to track features. Reinders *et al.* [5] describe a tracking technique which utilizes feature attributes. The feature data consists of basic attributes such as position, size, speed, orientation, etc. For each set of attributes, a number of correspondence functions can be tested, resulting in a correspondence factor. This factor makes it possible to quantify the goodness of the match between two features in successive time frames. Since the algorithm uses feature data, it is efficient in execution.

The tracking algorithm proposed is based on a simple assumption. Features evolve consistently and that they have attributes which can be used to track them. Some examples are listed below.

3.3.1 The Sense of Rotation

A simple example of one such characteristic from a fluid-dynamics perspective would be the sense of rotation of a vortex in a flow field. Vortices usually maintain their sense of direction, i.e., clockwise or counter clockwise, over time as they evolve, unless there is some shearing flow changing the sense of rotation. If it is known beforehand that it is two-dimensional vortices that we are comparing, then it would help to store the sense of rotation of the vortex as a feature attribute. The problem is more complicated in three dimensions. This flag can be used to reduce redundant comparisons for evaluating a best match. This concept is illustrated in the figure 3.9. In figure 3.9, the closed boxes represent sub-domain having features labeled *(a,b)* in time step *t1* and *(1,2)* in time step *t2*. Typically, the comparisons that would need to be done are *(a-1,a-2,b-1,b-2)*. However, as we do know from the rotation flag, which signifies the sense of rotation, that *a-2* and *b-1* are superfluous. This accelerates the process. There are other attributes to examine apart from the sense of rotation.

Another salient attribute of a feature is its strength. The size/mass of the feature may be representative of its strength. If we store the information about how "strong" a feature is, we can assume a "strength" gradient to exist and hence conclude that the strength cannot increase/decrease beyond a certain limit between adjacent time steps. This further reduces the number of redundant comparisons. The figure 3.10 illustrates this concept. As in figure 3.9, comparisons *a-2* and *b-1* are made superfluous.

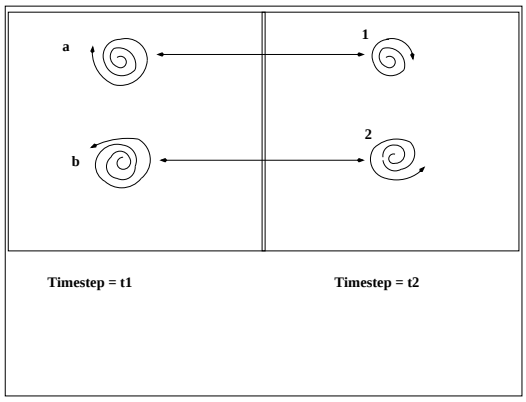

Figure 3.9: Using the sense of rotation as an attribute.

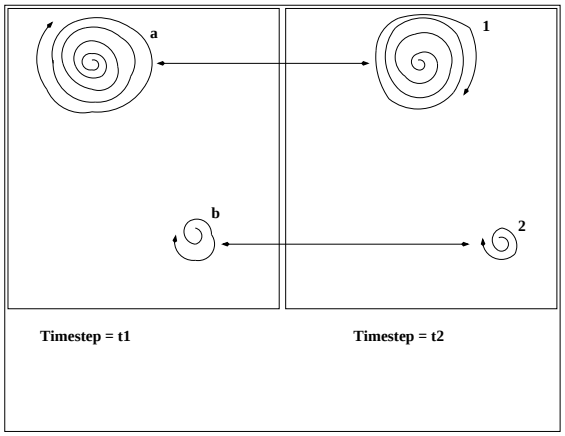

Figure 3.10: Strength Based Comparison.

3.3.2 Evaluating Correspondence

In order to detect a correspondence between candidate features, a series of correspondence functions C_{func} are evaluated for each pair. Each of the functions is accompanied by a tolerance T_i and a weight W_i. The value attained by C_{func} is kept within $[0, 1]$ if the deviation is kept within tolerance T_i. It is given a negative value otherwise. This is shown below.

$$C_{func} = \begin{cases} 1 & \text{Exact Match} \\ 0 & \text{Limit of Tolerance} \\ < 0.0 & \text{No Match} \end{cases}$$

The total correspondence between any two pair of features being compared is computed as a weighted sum of all the contributing correspondence functions. The result is stored as C_{corr}. If N_{func} is number of contributing functions, then the correlation is as defined below.

$$C_{corr} = \frac{\sum_{i=1}^{N_{func}} C_{func_i} W_i}{\sum_{i=1}^{N_{func}} W_i}$$

The number of correspondence functions, N_{func}, depends on the attribute sets of the features under study. An event graph over time for all the features is plotted. This is a mapping of how the features evolve over time. The correspondence functions are used to determine a confidence index for a given path of evolution of a feature. Reinders *et al.* [5] define a parameter τ, called the growth factor, as the length of the minimal path of evolution. In order to track a feature, the evolution of the feature in the event graph has to be followed. As the event graph is a directed graph, several paths may exist. In order to choose the best path, a

confidence parameter C_{conf} is calculated over the constituent *edges*.

$$C_{conf}(path) = 1 - e^{\frac{t}{\tau}}$$

$$t = \sum_{i=1}^{edges} C_{corr_i}$$

More details can be found in [5]. An example of some simple attributes of features described by [5] that can be used for tracking is shown in table 3.3.2.

Correspondence Function	Correspondence factor	Consistency Rule
$\frac{\|V_p - V_c\|}{max(V_p, V_c)} \leq T_{Vol}$	$C_{vol} = 1 - \frac{\frac{\|V_p - V_c\|}{max(V_p, V_c)}}{T_{Vol}}$	consistent growth
$dist(p, c) \leq T_{pos}$	$C_{pos} = 1 - \frac{dist(p,c)}{T_{pos}}$	consistent speed
$1 - \|cos(\angle(\bar{p}, \bar{c}))\| \leq T_{angle}$	$C_{angle} = 1 - \frac{1 - \|cos(\angle(\bar{p}, \bar{c}))\|}{T_{angle}}$	consistent rotational speed

In the expressions in table 3.3.2, the subscript p stands for the feature in time step t and c for time step $t + \Delta t$. Attributes listed are the volume, rotational speed and position.

The attribute-based technique can be used if the phenomena under study assigns to the features, data such as size/mass that are easy to capture. This would greatly improve tracking time. However it is not possible to apply this technique on all classes of features.

CHAPTER IV

PREDICTIVE FEATURE TRACKING

Chapter III discussed the various techniques used for tracking features in time varying datasets. In this chapter, we present an enhancement to improve upon those techniques by incorporating predictive techniques.

4.1 Predictive Techniques

Motion correspondence has a number of applications in computer vision including motion analysis, object tracking, surveillance, etc. Several techniques have been mentioned in [9] [10]. These include, the *nearest neighbor* model, which assumes that features move as little as possible from time t to $t + \Delta t$. This is equivalent in concept to the method of Silver [7] for feature tracking. Sethi and Jain [11] suggest a *smooth-motion* model which assumes that the velocity magnitudes vary in magnitude and direction gradually. A heuristic model is given in [11]. The *proximity-uniformity* model by Rangarajan and Shah [12] assumes little motion in addition to constant speed.

The rationale behind the current effort is to combine the generalized tracking algorithm and attribute-based techniques to develop a technique for predicting the physical location and orientation of a feature and exploit this additional

information to improve our confidence in the tracking process. Implicitly, we are employing additional information for conducting comparisons between features.

4.2 Motion of Features

There are three broad classes of motion that a feature can undergo.

1. Translation.
2. Rotation.
3. Deformation.

This study attempts to estimate the translation and rotation about the centroid of a feature for prediction. In essence, we are saying that the salient attributes of a feature are its translational motion and rotational motion. No attempt is made to estimate the deformation of the feature. As stated above, the general tracking algorithm assumes that features evolve consistently over all time. The technique we study assumes the following:

1. That the features evolve consistently over a set of three consecutive time steps.
2. Over this period, the features undergo rigid translation and uniform rotation about its centroid.

If there are two time steps, then there is no way to predict anything about the features in the second time step based on the first one alone, and the basic tracking algorithm is employed. If there are three time steps, the information from the first two time steps can be used to predict the physical location and angular orientation of the feature in the third time step. If there are four or more time steps, the

following is proposed. The data set is sampled at three consecutive time-steps to predict the physical location of the feature in the fourth time-step. This technique is applied repeatedly over the second, third and fourth time-steps to predict the fifth. The fifth, sixth and seventh time-steps are used to predict the eighth time step, and so on. The figure 4.1 makes this clear.

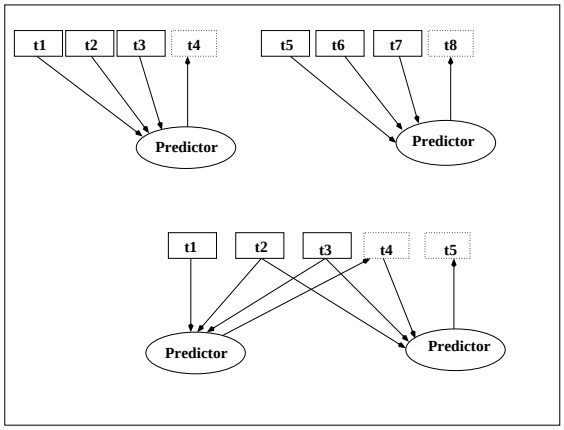

Figure 4.1: The Predictive Scheme

4.3 The Issue of Various Grid Types

The general tracking algorithm proposed by Silver [7] works for an unstructured mesh. However, implementing a technique to preserve and translate a shape from one location on the grid to another is difficult to implement on unstructured meshes. The issue persists for structured, non-cartesian meshes also. A simple solution is proposed.

An ellipse that encompass the feature is made as representative of the feature. The shape of an ellipse is chosen over a rectangle/square shape because this study focuses on studying vortices as features and an ellipse would be a good approximation for the shape. The major axis of the ellipse is taken to be the principal axis and the minor axis perpendicular to the major axis. Let (x_i, y_i)

represent the set of n nodes that define a feature. The principal axis can be defined as the line passing through the feature such that the sum of the distances of all the points to that line is a minimum as shown in equation 4.1

$$\sum_{i=0}^{n}(y_i - f(x_i))^2 = minimum. \tag{4.1}$$

The slope of such a line is given by equation 4.2

$$\tan\theta = \frac{(\sum y)(\sum x^2) - (\sum x)(\sum xy)}{(n\sum x^2) - (\sum x)^2}. \tag{4.2}$$

The equation of any ellipse centered at the centroid (x_{cen}, y_{cen}) of a feature is

$$\frac{(x - x_{cen})^2}{a^2} + \frac{(y - y_{cen})^2}{b^2} = 1. \tag{4.3}$$

In order to completely define the ellipse encompassing the feature, two points that lie on the ellipse are needed. The two points are chosen to be the farthest points from the major and minor axes. Figure 4.2 illustrates the process. Let these points

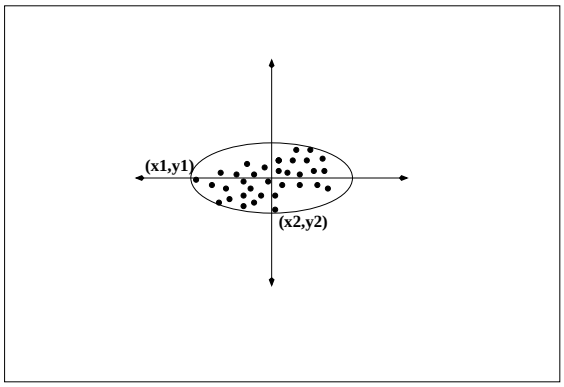

Figure 4.2: The Ellipse Representation

be (x_1, y_1) and (x_2, y_2). Substituting the values $x = x_1, y = y_1$ and $x = x_2, y = y_2$

gives us two equations

$$\frac{(x_1 - x_{cen})^2}{a^2} + \frac{(y_1 - y_{cen})^2}{b^2} = 1 \tag{4.4}$$

$$\frac{(x_1 - x_{cen})^2}{a^2} + \frac{(y_1 - y_{cen})^2}{b^2} = 1. \tag{4.5}$$

Solving for a and b gives us

$$a = \sqrt{\frac{|(x_1 - x_c)^2(y_2 - y_c)^2 - (x_2 - x_c)^2(y_1 - y_c)^2|}{|(y_2 - y_c)^2 - (y_1 - y_c)^2|}} \tag{4.6}$$

$$b = \sqrt{\frac{|(y_1 - y_c)^2(x_2 - x_c)^2 - (y_2 - y_c)^2(x_1 - x_c)^2|}{|(x_2 - x_c)^2 - (x_1 - x_c)^2|}}. \tag{4.7}$$

From basic geometry, it is known that the foci's of an ellipse, centered at the origin, are on the major axis, a distance c away along the positive and negative x-direction. The value of c is given by

$$c = \sqrt{a^2 - b^2}.$$

An ellipse which has its major axis slope of $\tan\theta_{major}$ and passing through (x_{cen}, y_{cen}) has the equation

$$y - y_{cen} = \tan\theta_{major}(x - x_{cen}). \tag{4.8}$$

The x-coordinates of the foci's are $x_{cen} \pm c\cos\theta_{major}$. Substituting

$$x_{focus_1} = x_{cen} + \frac{c}{\sqrt{1 + \tan^2\theta_{major}}} \tag{4.9}$$

$$x_{focus_2} = x_{cen} - \frac{c}{\sqrt{1 + \tan^2\theta_{major}}} \tag{4.10}$$

gives us the values of the y-coordinates of the foci's

$$y_{focus_1} = y_{cen} + \frac{c \tan \theta_{major}}{\sqrt{1 + \tan^2 \theta_{major}}} \tag{4.11}$$

$$y_{focus_2} = y_{cen} - \frac{c \tan \theta_{major}}{\sqrt{1 + \tan^2 \theta_{major}}}. \tag{4.12}$$

The next step in the process of generating an ellipse-like representation of the feature is to make all nodes that lie within the defined ellipse as significant. From the definition of an ellipse, it is known that an ellipse is described as the loci of a point P which moves such that the sum of the distance from any two fixed points (the foci's, say F_1 and F_2) is a constant $2a$.

$$|PF_1| + |PF_2| = 2a \tag{4.13}$$

All points P that lie within this ellipse, would have the property

$$|PF_1| + |PF_2| \leq 2a. \tag{4.14}$$

Equation 4.14 is used to switch on all the nodes within the ellipse as significant. The application of predictive techniques for feature tracking follows.

4.4 The General Approach

The general idea is to use techniques for predicting the physical location and angular orientation of a feature in the fourth time step based on its location at the previous three time steps. The predicted feature is stored as an ellipse-like representation. In order to encode translation, it is assumed that the feature undergoes uniform acceleration. It may be noted that the rotation prediction

helps us by giving more overlap and therefore more confidence on the predicted match. The process can be described as follows.

1. The location of the centroid of the feature is predicted based on its position in previous time-steps.

2. An ellipse-like representation of the feature (based on its shape in the previous time-step) is placed at this location.

3. The major axis of the ellipse is rotated by an angle predicted based on the orientation of the feature in previous time-steps.

For the purpose of analysis in subsequent sections, let us assume that (x_1, y_1),(x_2, y_2) and (x_3, y_3) are positions of the centroid of a feature in time steps t, $t + \Delta t$ and $t + 2\Delta t$. Let the centroid of the feature to be predicted be (x_p, y_p). The various techniques used are discussed in subsequent sections.

4.5 Translation

If a particle is at point (x_0, y_0), undergoing motion with an initial velocity $\overrightarrow{u_0}$, with a constant acceleration $\overrightarrow{a_0}$, then the predicted position of the particle after a time interval Δt is given by

$$x_p = x_0 + u_x \Delta t + \frac{1}{2} a_x \Delta t^2 \tag{4.15}$$

$$y_p = y_0 + u_y \Delta t + \frac{1}{2} a_y \Delta t^2. \tag{4.16}$$

It is assumed that a constant time step, Δt, is used.

Predicting the location of a feature in a future time-step based on two previous time-steps is now discussed. It is not possible to predict acceleration by using two time-steps. Assume that the feature is at location (x_1, y_1) at time t_1 and at location (x_2, y_2) at time t_2, the predicted location (x_p, y_p) at time t_3 can be computed using

$$u_x = \frac{x_2 - x_1}{\Delta t} \tag{4.17}$$

$$u_y = \frac{y_2 - y_1}{\Delta t} \tag{4.18}$$

$$a_x = a_y = 0 \tag{4.19}$$

which implies

$$x_p = x_2 + \frac{x_2 - x_1}{\Delta t}\Delta t \tag{4.20}$$

$$y_p = y_2 + \frac{y_2 - y_1}{\Delta t}\Delta t \tag{4.21}$$

and yields

$$x_p = 2x_2 - x_1 \tag{4.22}$$

$$y_p = 2y_2 - y_1. \tag{4.23}$$

The case of prediction using three previous time-steps is now considered. A feature's initial velocity can be calculated based on the location of its centroids at

time steps t_1,t_2 and t_3 as follows

$$u_x = \frac{3x_3 - 4x_2 + x_1}{2\Delta t} \tag{4.24}$$

$$u_y = \frac{3y_3 - 4y_2 + y_1}{2\Delta t}. \tag{4.25}$$

The acceleration is then computed as

$$a_x = \frac{x_3 - 2x_2 + x_1}{\Delta t^2} \tag{4.26}$$

$$a_y = \frac{y_3 - 2y_2 + y_1}{\Delta t^2}. \tag{4.27}$$

Substituting values for a_x and a_y in equation 4.15

$$x_p = x_3 + \frac{(3x_3 - 4x_2 + x_1)}{2\Delta t}\Delta t + \frac{1}{2}\frac{(x_3 - 2x_2 + x_1)}{\Delta t^2}(\Delta t)^2 \tag{4.28}$$

$$y_p = y_3 + \frac{(3y_3 - 4y_2 + y_1)}{2\Delta t}\Delta t + \frac{1}{2}\frac{(y_3 - 2y_2 + y_1)}{\Delta t^2}(\Delta t)^2 \tag{4.29}$$

$$x_p = 3x_3 - 3x_2 + x_1 \tag{4.30}$$

$$y_p = 3y_3 - 3y_2 + y_1. \tag{4.31}$$

4.6 Rotation

In order to capture rotation, it is assumed that the difference in the orientation of the principal axis gives the angle by which the feature has rotated. Recall that the principal axis is defined as the line passing through a feature such that the sum of distances of all nodes within the feature to the line is a minimum as defined in equation 4.1. The slope of such a line is $\tan\theta$, where $\tan\theta$ is defined by equation 4.2. Let θ_1, θ_2 and θ_3 be the orientations of the feature in the three time steps.

Following in the same lines as equation 4.15 gives us

$$\theta_p = \theta_3 + \frac{3\theta_3 - 4\theta_2 + \theta_1}{2\Delta t}\Delta t + \frac{1}{2}\frac{\theta_3 - 2\theta_2 + \theta_1}{(\Delta t)^2}\Delta t^2 \tag{4.32}$$

When the angular acceleration is zero, the third term in equation 4.32 is zero. For the case of features rotating at constant angular velocity, equation 4.32 yields

$$\theta_p = 2\theta_2 - \theta_1. \tag{4.33}$$

If there is angular acceleration, equation 4.32 yields

$$\theta_p = 3\theta_3 - 3\theta_2 + \theta_1. \tag{4.34}$$

As is the case with predicting translation based on two previous time-steps, it is not possible to predict the angular acceleration. This value θ_p is used to evaluate the orientation of the principal axis in the fourth time step.

CHAPTER V

RESULTS AND DISCUSSION

The predictive feature tracking algorithm discussed in the previous chapter was tested on an artificially generated data set which had a single accelerating feature. Yet another dataset had a feature undergoing approximately uniform angular rotation and displacement. The artificial dataset is included to demonstrate that additional overlap can be obtained using predictive techniques. The algorithm was also tested on a real data set for viscous flow past a cylinder. In this case, the cylinder sheds vortices and the vortices are convected downstream by the flow. Plots of overlap values obtained using the prediction based on two previous time-steps and three previous time-steps are plotted. As discussed in chapter II, the swirl parameter is used to detect the vortices and the region growing algorithm is used to segment the field into regions of interest.

5.1 Artificial Data Sets

A set of artificial time varying data was generated as a preliminary test case. The grid dimensions were 100×100. The data set had a moving geometric figure (a square) that represented a feature. The velocity and acceleration of the object was predefined. In order to demonstrate the algorithm, the object is given a size of $10x10$ nodes, and an acceleration of $2 nodes/sec^2$. Those values were chosen so there would be no physical overlap in adjacent time-steps after the 4^{th} time-

step. The predictive module is then used to estimate the feature's location in the next time step. The figures 5.1, 5.2, 5.3, 5.4 and 5.5 illustrate the process. The overlapping areas are shown in green. The square in the earlier time-step is shown in red and the later time-step is shown in yellow. The position of the ellipse-like representation of the feature using the predictive techniques is rendered. The status bars on the right of snapshot show the degree of overlap as computed by equation 3.1. The overlap values obtained with the predictive techniques are shown in table 5.1. The overlap values indicate overlap with the previous time step.

Time	Basic Overlap	prediction using 2 time-steps	prediction using 3 time-steps
2	36.00	56.67	-
3	16.00	56.67	100
4	4.0	56.67	100
5	0	56.67	100
6	0	56.67	100
7	0	56.67	100

Table 5.1: R values for feature ID 0,undergoing uniform acceleration, using basic overlap and prediction using information from the previous two and three time steps

As was stated earlier, the feature has finite acceleration. This causes the basic overlap to become zero at the fourth time-step. It can also be seen in table 5.1 that the prediction using three previous time-steps gives a better estimate than

using two previous time-steps. This is because it is not possible to predict the acceleration of a feature using its location at two previous time-steps. However, the use of two previous time-steps for prediction is better than basic overlap as it gives a better estimate, if not the best, of the location of a feature in a future time-step.

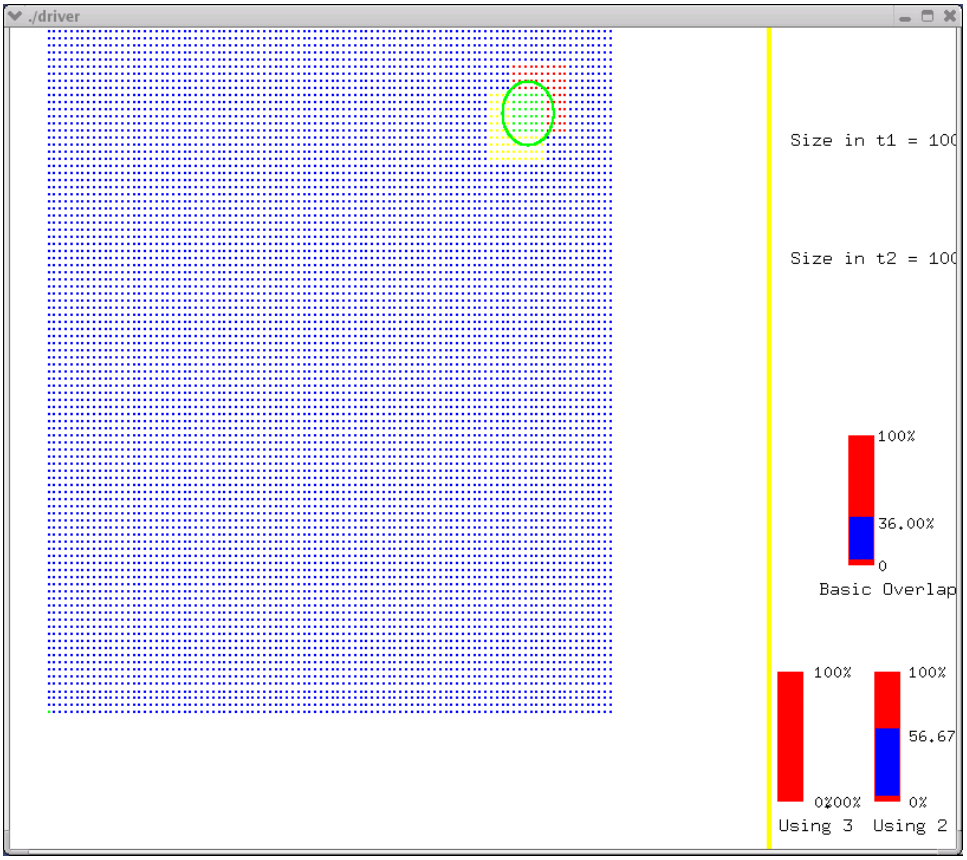

Figure 5.1: Snapshot of the Artificial Dataset using predictive tracking (time step = 1)

The rotation prediction routines are tested on artificially generated data. This dataset has a rectangle undergoing coupled rotation and translation. The rectangle in yellow in the later time-step and red in earlier time-step. The green area shows the area of overlap detected. The ellipse-like representation of the feature

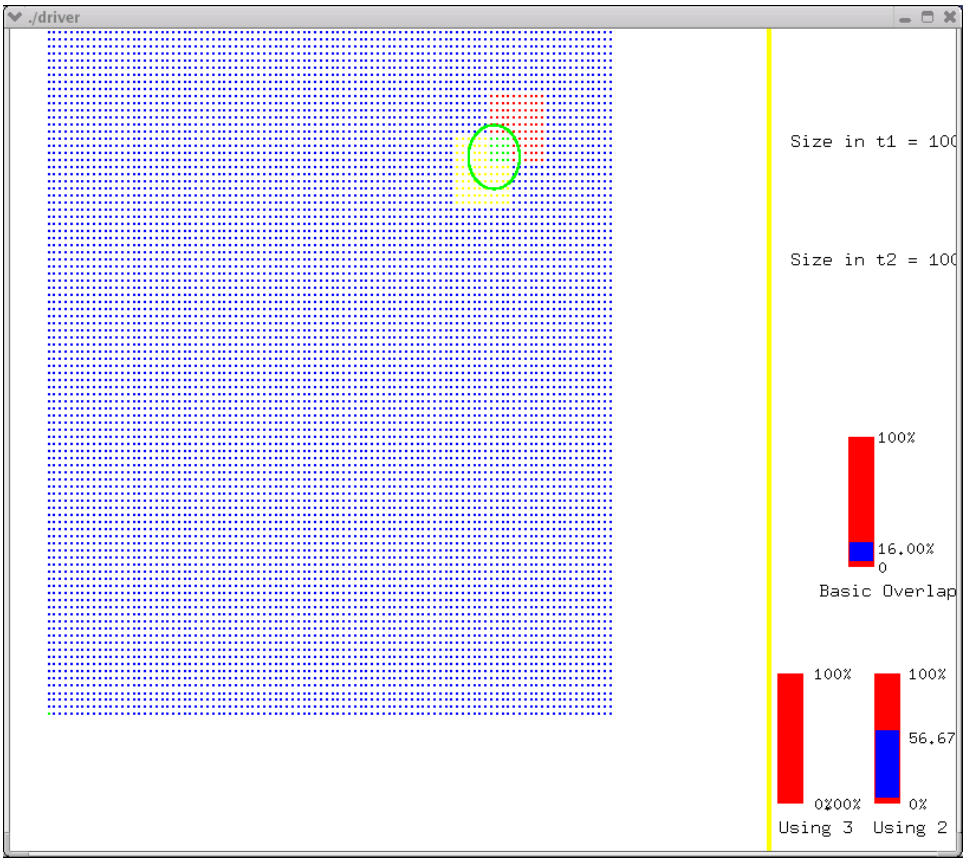

Figure 5.2: Snapshot of the Artificial Dataset using predictive tracking (time step = 2)

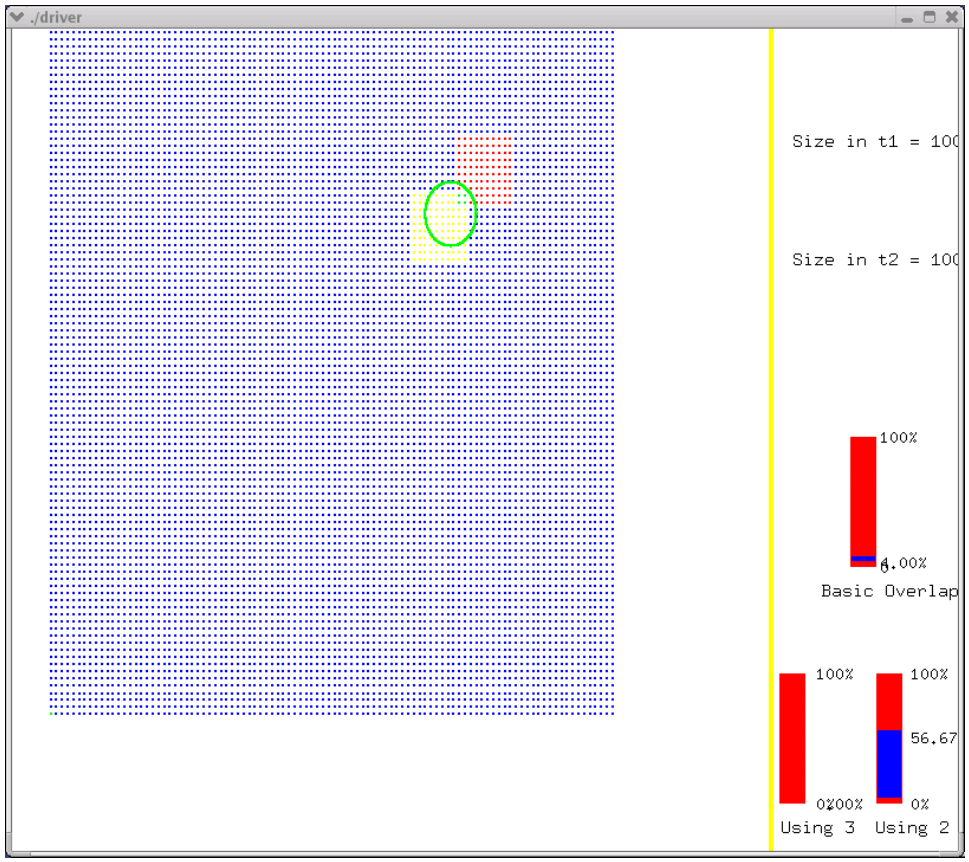

Figure 5.3: Snapshot of the Artificial Dataset using predictive tracking (time step = 3)

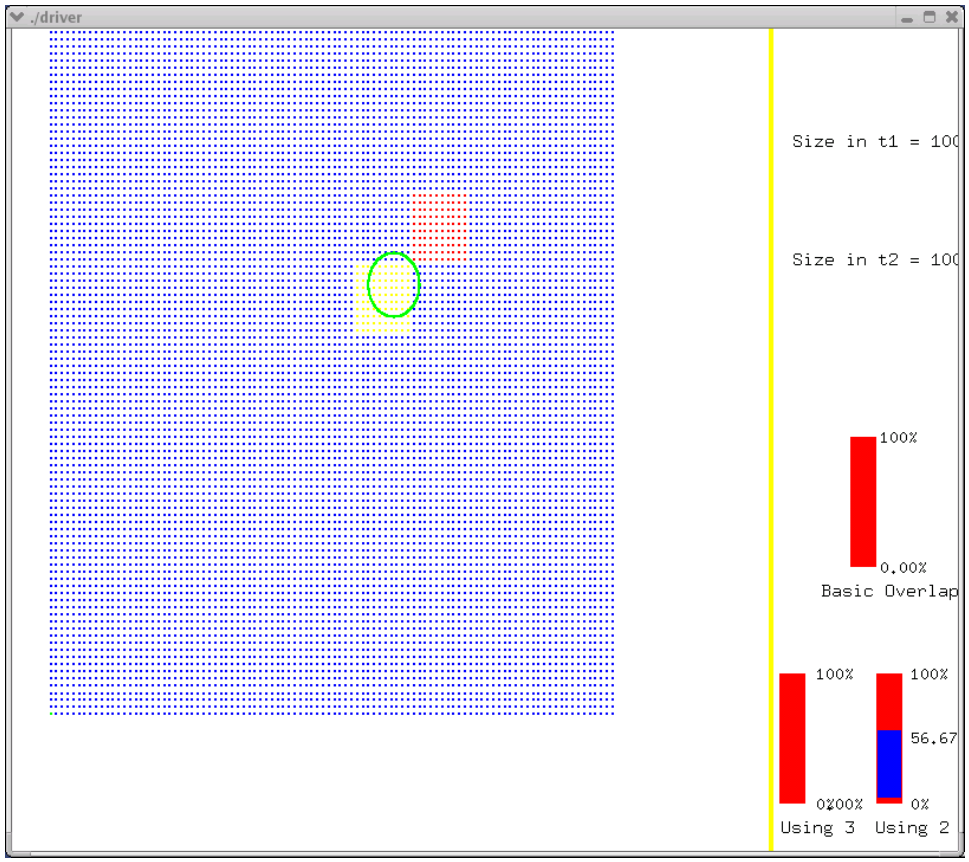

Figure 5.4: Snapshot of the Artificial Dataset using predictive tracking (time step = 4)

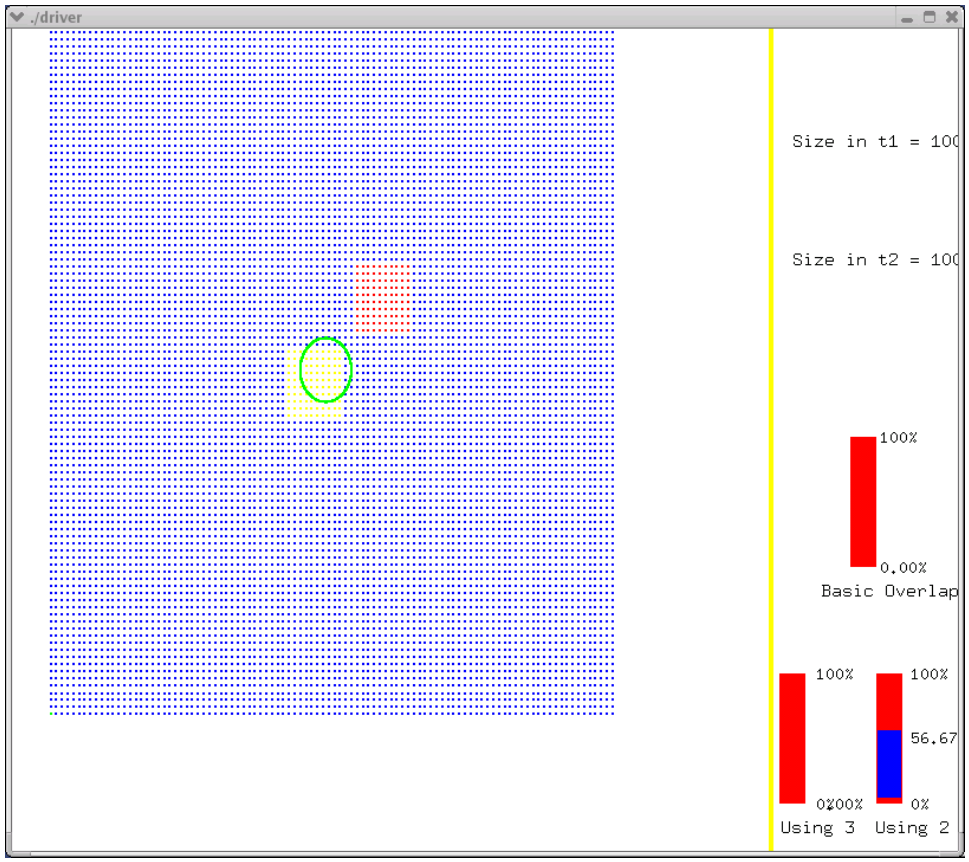

Figure 5.5: Snapshot of the Artificial Dataset using predictive tracking(time step = 5)

shows how the use of predictive techniques increased overlap. Table 5.2 shows the improvement in the overlap obtained using predictive techniques.

Time	Basic Overlap	prediction using 2 time-steps	prediction using 3 time-steps
2	36.12	89.10	-
3	35.99	89.22	88.98
4	36.02	88.42	88.35
5	36.06	88.99	89.51
6	36.09	88.91	89.07
7	36.09	89.97	90.28
8	35.90	92.06	90.53

Table 5.2: R values for feature ID 0,undergoing uniform rotation, using basic overlap and prediction using information from the previous two and three time steps

A rectangle, is rotated with uniform angular velocity, and translated, to simulate a coupled translating rotating feature. As the feature rotates and translates, the basic overlap between time-steps is small as shown in table 5.2. However, the use of two previous time-steps to prediction the location and orientation of the feature in a later time-step has increased the overlap. The use of the previous three time-steps has yielded similar results as the feature has no angular acceleration. The increase in overlap has increased the confidence in the match obtained for tracking the feature.

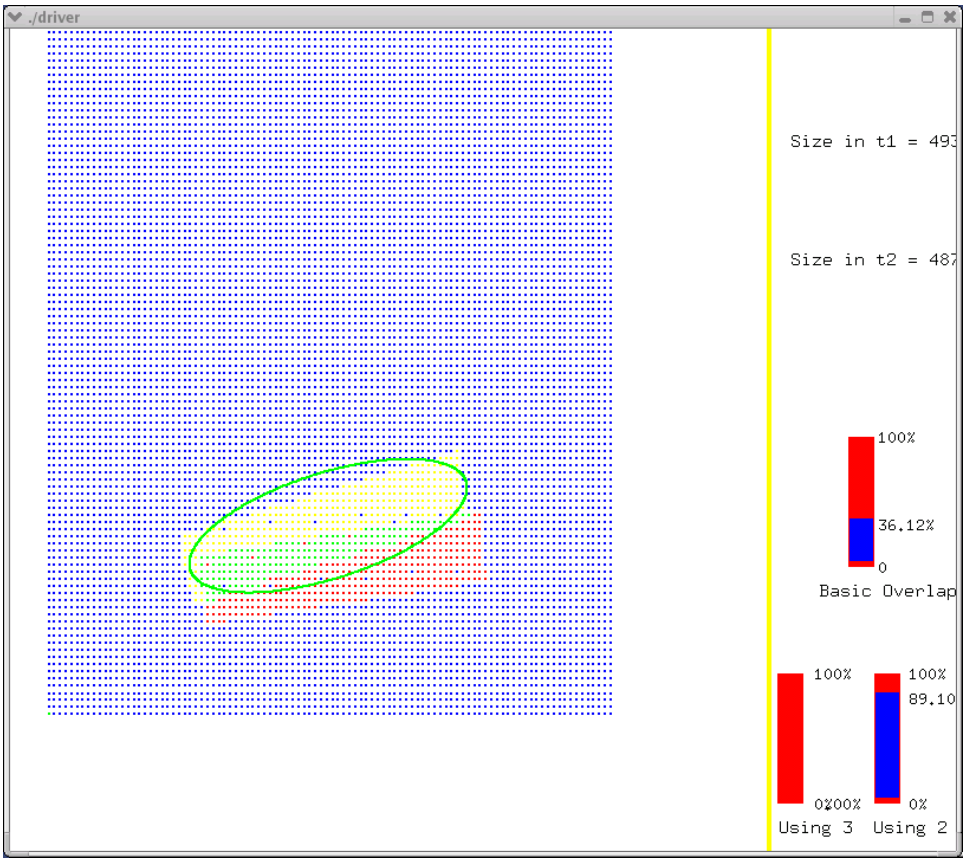

Figure 5.6: Snapshot of overlap with rotation prediction (time step = 2)

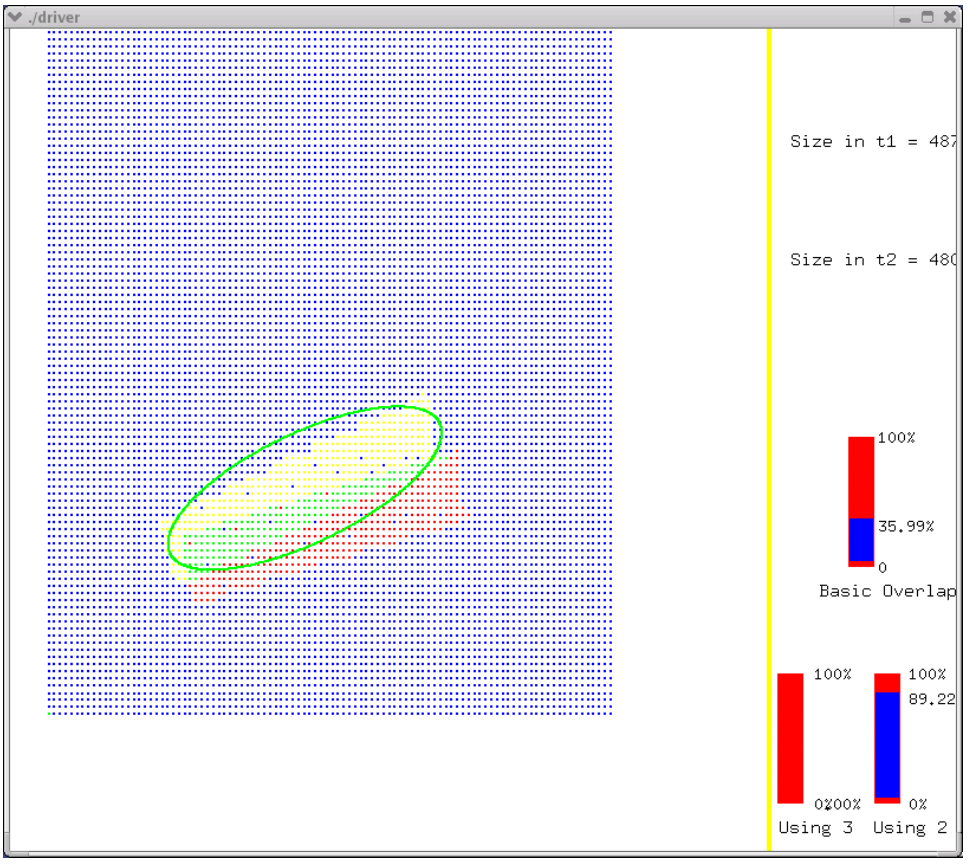

Figure 5.7: Snapshot of overlap with rotation prediction (time step = 3)

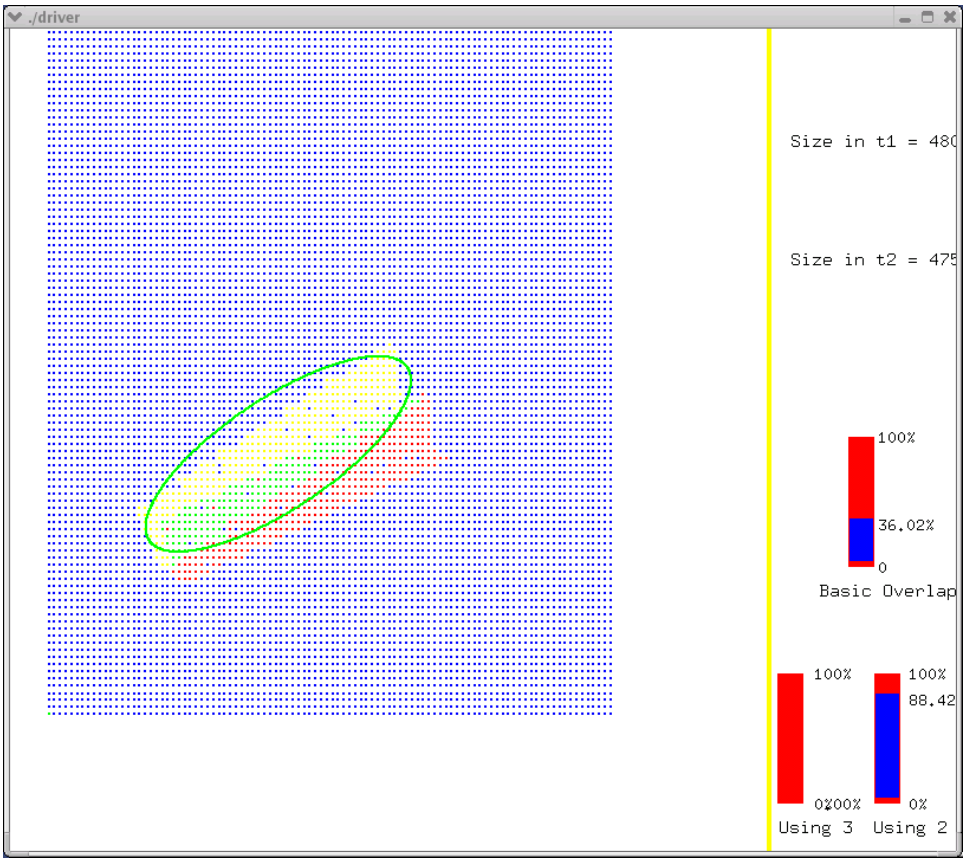

Figure 5.8: Snapshot of overlap with rotation prediction (time step = 4)

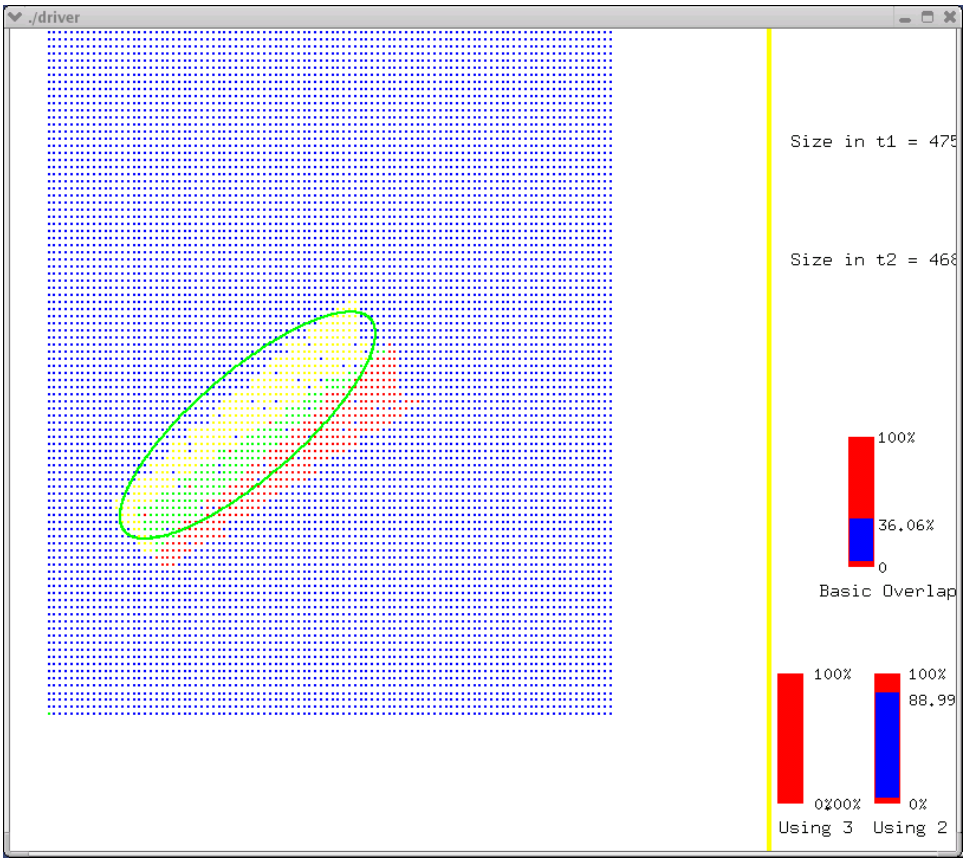

Figure 5.9: Snapshot of overlap with rotation prediction (time step = 5)

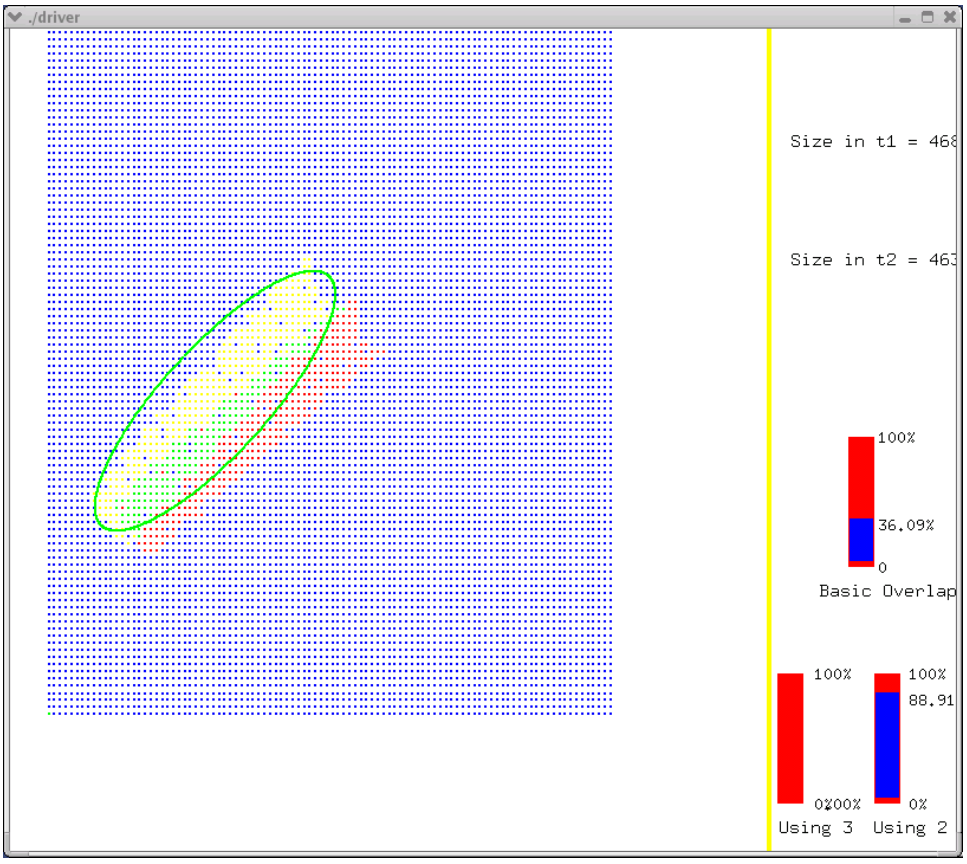

Figure 5.10: Snapshot of overlap with rotation prediction (time step = 6)

5.2 Flow Past Cylinder

A time-varying simulation of viscous flow past a cylinder was downloaded from the Internet at [13]. The cylinder starts shedding vortices and these vortices are convected downstream with the flow. The grid used for this simulation is shown in figure 5.11. The process of representing features, as described in chapter IV, is implemented. The original data had dimensions $64 \times 64 \times 32$. The z-slice at 20 was chosen.

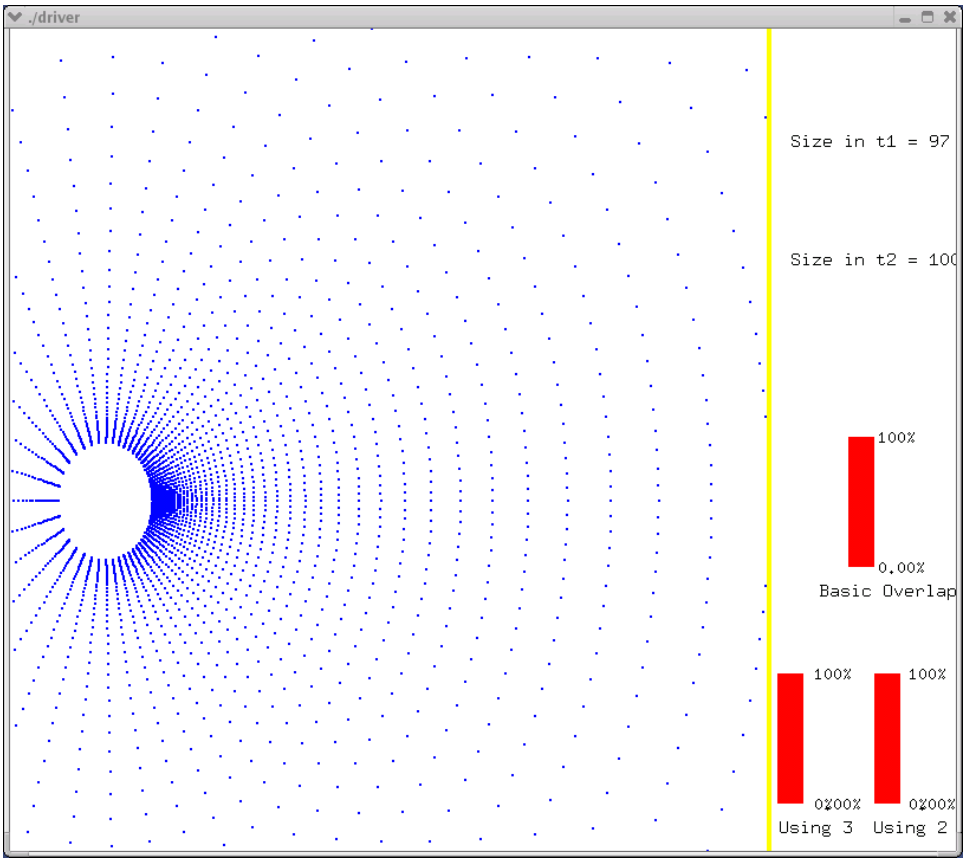

Figure 5.11: Snapshot of the Grid used for flow past a cylinder

The swirl values are computed at all the nodes. Nodes with τ values greater than a specified threshold are marked as significant. The nodes are segmented into features by the *region growing* technique mentioned in Chapter III. The time

interval of sampling is varied to demonstrate the variation of the overlap computed using the basic overlap technique (Silver [7]), prediction using only two previous time steps, and prediction using three time steps. The time interval is varied from one to ten. The variation of the overlap parameter is shown in figures 5.12 and 5.13.

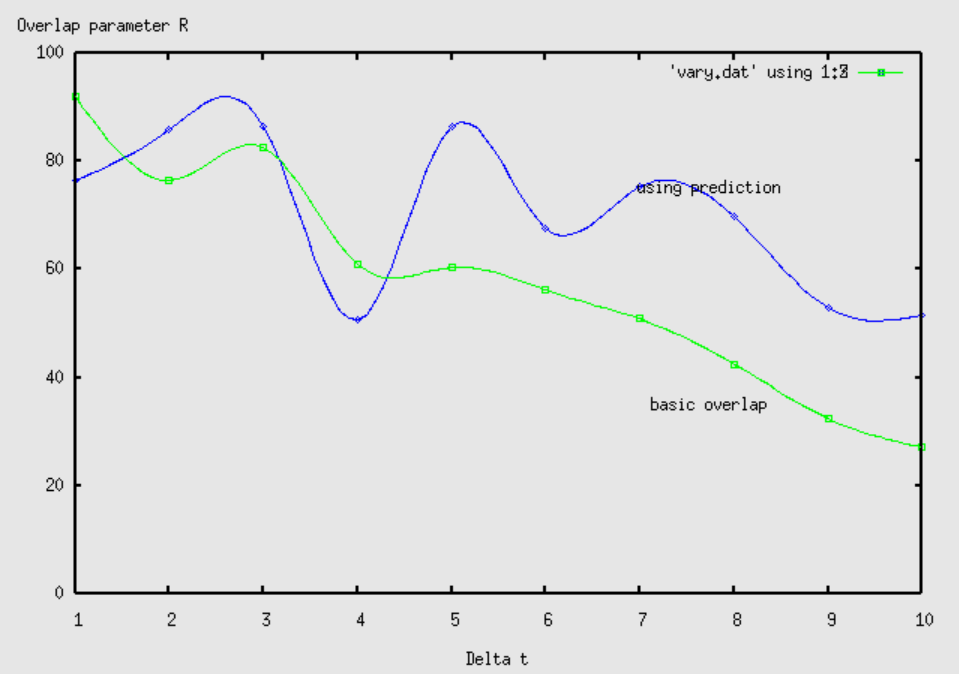

Figure 5.12: Plot of R value for $\Delta t = 1...10$ for the 3^{rd} time step using 2^{nd} and 1^{st}

The lower green line indicates the degree of area overlap obtained using basic overlap techniques. The upper blue line indicates the increment in overlap obtained using predictive techniques. For the purpose of this study, one feature is kept visible and tracked. This is chosen to be a vortex shed from under the cylinder. It is observed that this vortex travels fairly linearly, and is well suited as a test case for this study. Figures 5.14-5.16 illustrate this process.

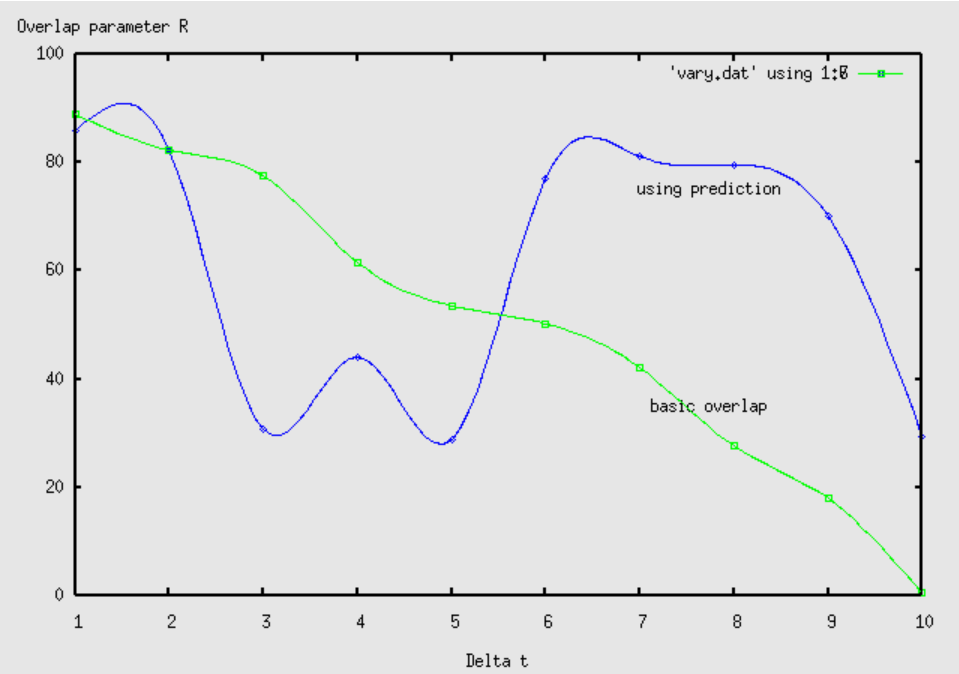

Figure 5.13: Plot of R value for $\Delta t = 1...10$ for the 4^{th} time step using 3^{rd}, 2^{nd} and 1^{st}.

Use of the predictive technique has helped develop a better overlap value. The figures 5.12 and 5.13 also indicate that we can afford to subsample the time variant data set by not compromising overlap. This could help save memory.

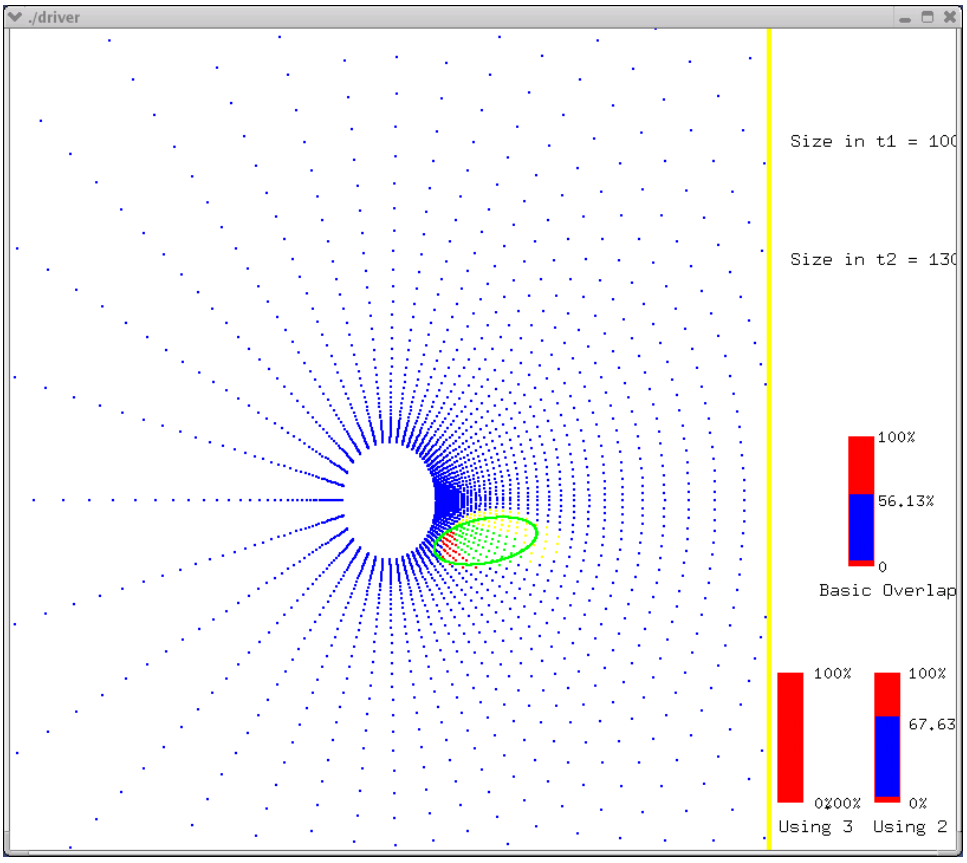

Figure 5.14: Snapshot of the use of predictive techniques for time $t = 37$ using features from $t = 31$ and $t = 25$

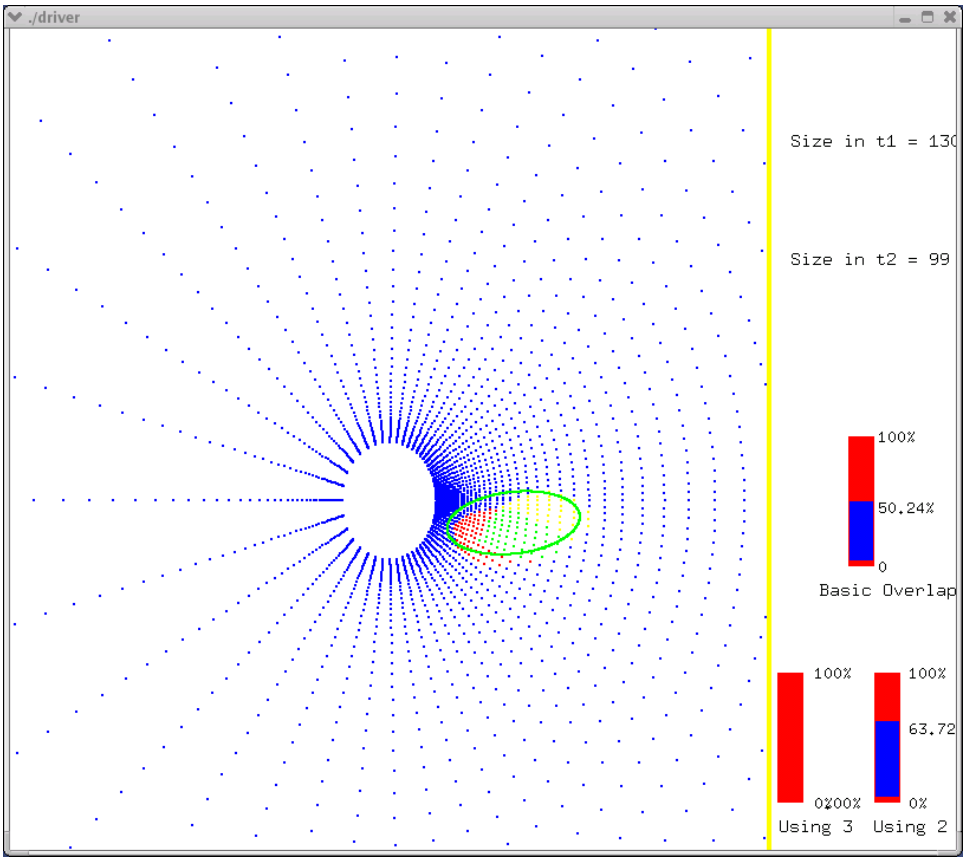

Figure 5.15: Snapshot of the use of predictive techniques for time $t = 43$ using features from $t = 37$ and $t = 31$

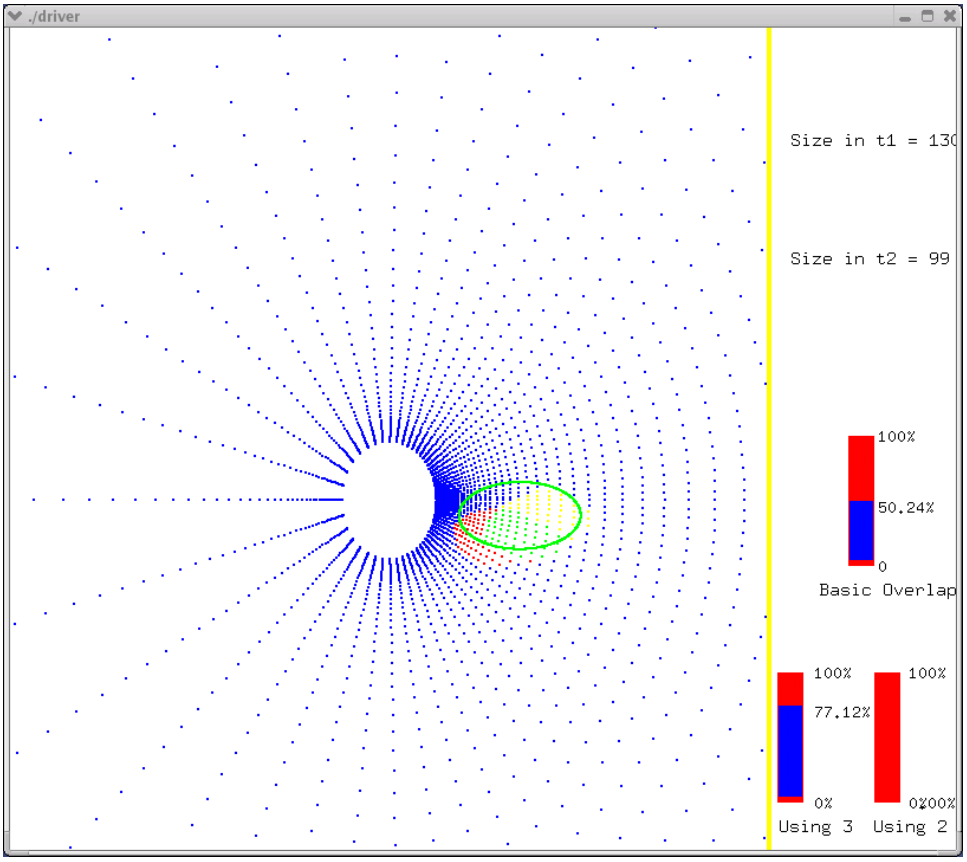

Figure 5.16: Snapshot of the use of predictive techniques for time $t = 43$ using features from $t = 37$, $t = 31$ and $t = 25$

CHAPTER VI

CONCLUSIONS AND FUTURE WORK

The present work deals with predictive feature tracking techniques for time varying data sets. The general tracking algorithm was tested on an artificial data set with features undergoing both translation and rotation. When the general tracking algorithm was applied to the artificial data set, it was noticed that some features, being the same, did not show sufficient overlap. The new algorithm helped track such features by employing predictive techniques. The translation estimation techniques were effectively used to predict the centroid of a feature in a future time step. The rotation estimation techniques helped increase the confidence in the match by increasing the overlap. The algorithm was also tested on a time varying data set of a flow past a cylinder. The figures 5.12 and 5.13 indicate that we could subsample the data set while loading it into memory as the predictive techniques help increase the overlap. In general, prediction based on two previous time-steps does a better job of estimating the location of a feature in a future time-step than using three previous time-steps. It is also of interest to note the results in table 5.1. When equation 4.22 is used to predict the location of a feature in a future time-step, there is a constant "phase-lag" between the predicted location and the actual location. This can be verified mathematically. Use of previous two time-steps for prediction is a viable option in datasets which have relatively slow moving features. However the technique has

certain limitations. The algorithm handles all translation as approximately linear motion. The case of a feature, moving along a complex random curve, is difficult to track. An ellipse-life representation of a feature was used to predict overlap. The feature was assumed to have a shape which could be contained within an ellipse. This need not always be the case. It is also assumed that the feature rotates about its centroidal axis. This is another limitation of the algorithm.

In summary, this technique can be used effectively on data sets which have a few features, undergoing rapid translation/rotation. It may not be effective in cases when there are a large number of features undergoing various forms of motion.

6.1 Future Work

The idea of translating a shape based on its shape to a future time-step and checking to see overlap is difficult to implement on unstructured grids. A means for parameterizing shapes on unstructured grids, and effectively reconstructing them at any location, is needed. The idea of using ellipse-like representations of features to represent their shape is one way of predicting the size of a feature based on its size in the previous two/three time steps. Yet another approach could be to define control points around feature boundary and use a curve fit as a boundary of that feature. The movement of these control points can be used to predict the shape of the feature in a future time step. The case of preserving the characteristics of a feature undergoing rotation, shape change, and translation is difficult to capture on an unstructured grid. The application of translational/rotational predictive techniques for unstructured meshes which preserve the shape of the feature is also challenging. This is the scope for future work.

REFERENCES

[1] R. M. J. F. D. T. W. Schroeder and B. K. Soni, "Evita - efficient visualization and interrogation of tera-scale data," *R. L. Grossman et. al, eds, Kluwer Academic Publishers*, pp. 257–279, 2001.

[2] O. Faugeras, "3d computer vision," *The MIT press, Cambridge Massachusetts*, 1993.

[3] Q. C. J.K. Aggarwal, "Human motion analysis: A review," *Proc. IEEE Computer Society Workshop on Motion of Non-Rigid and Articulated Objects*, pp. 90–102, 1997.

[4] X. Wang, "Visualizing and tracking features in 3d time varying datasets," 1999.

[5] F. R. F. H. Post and H. J. Spoelder, "Attribute based feature tracking,"

[6] T. D. Berdahl C.H., "Eduction of the swirling structure using the velocity gradient," *Journal of Computational Physics*, vol. 37, pp. 70–92, 1980.

[7] D. Silver, "Object oriented visualization," *IEEE Computer Graphics and Applications*, vol. 15(3), May 1995.

[8] M. Gao, "Data extraction and abstraction," *Masters Thesis, Rutgers University*, May 1992.

[9] M. R. C.J. Veenman and E.Backer, "Resolving motion correspondence for densely moving points," *Article*.

[10] I. Cox, "A review of statistical data association techniques for motion correspondence.," *Internation Journal of Computer Vision*, vol. 10(1), pp. 53–66, 1993.

[11] I. Sethi and R.Jain., "Finding trajectories of features in a monocular image sequence," *IEEE Transactions on Pattern Analysis and Machine Intelligence.*, vol. 9(1), pp. 91–97, 1987.

[12] K.Rangarajan and M.Shah, "Establishing motion correspondence," *CVGIP: Image Understanding*, vol. 24(6), pp. 91–97, July 1991.

[13] J. D. Set, "Tapered cylindrical unsteady flow," *http://www.nas.nasa.gov/Research/Datasets/datasets.html*.