FEATURE TRACKING OF 3D SCALAR DATASETS IN THE 'VISIT' ENVIRONMENT

BY ROHINI M PANGRIKAR

A thesis submitted to the

Graduate School—New Brunswick

Rutgers, The State University of New Jersey

in partial fulfillment of the requirements

for the degree of

Master of Science

Graduate Program in Electrical and Computer Engineering

Written under the direction of

Professor Deborah Silver

and approved by

_____

_____

_____

New Brunswick, New Jersey

October, 2008

ABSTRACT OF THE THESIS

Feature Tracking of 3D Scalar Datasets in the 'VisIt' Environment

by Rohini M Pangrikar

Thesis Director: Professor Deborah Silver

Visualization of 3D scientific time varying datasets is a challenging task due to the large amount of data to be processed. Such datasets can contain evolving patterns, visualization of which can give an intuitive interpretation of a dataset under study. The Vizlab at Rutgers has pioneered in the development of feature tracking of 3D scalar datasets by isolating features or region of interest and tracking them over subsequent time steps by using spatial matching. Once these features are extracted and tracked, their evolutionary information can be used for iso-surface visualization by color coding each feature such that evolving patterns are easy to follow.

This thesis presents the development of Feature Tracking of 3D scalar datasets in VisIt, a free interactive parallel visualization and graphical analysis tool for viewing scientific data. The implementation is divided into two modules. The first module, implemented as 'FeatureTrack' operator plug-in, performs feature extraction, quantification and stores iso-surface information for each timestep, then tracks features over subsequent timesteps. The second module, implemented in conjunction with the 'poly-file' reader plug-in, 'TrackPoly' operator plug-in and 'PolyDataPlot' plot plug-in, performs visualization. Separation of visualization from feature tracking enables 'selective enhanced visualization' so only features of interest are selected and tracked over time.

Table of Contents

Abstract ii

List of Tables vi

List of Figures vii

1. Introduction 1

2. Related Work 5

2.1. Importance of Feature Tracking in Visualization 5

2.1.1. What are 'Features'? 5

2.2. Existing 3D scalar data Feature Tracking algorithms 6

3. Overview of the Feature Tracking Algorithm Implementation 9

3.1. Feature Extraction and Quantification 10

3.2. Feature Tracking 11

3.3. Visualization 16

4. VisIt 18

4.1. What is VisIt? 18

4.2. VisIt Architecture 19

4.3. Visualizing 3D Dataset in VisIt 21

4.4. AVT in VisIt 22

4.4.1. Overview 23

4.4.2. Use of AVT in VisIt pipelines 23

4.5. vtkDataset 24

4.5.1. vtkDataset structural description 25

4.5.2. Types of vtkDataset 26

5. Feature Tracking and Visualization of 3D Scalar Datasets in VisIt Environment 28

5.1. Object Segmentation and Feature Tracking 28

5.1.1. Using the Software 29

Creation of the pipeline in GUI 29

Set attributes 31

Execution 33

Source Code 33

5.2. Visualization Using the Iso-surface Information 34

5.2.1. Using the Software 35

Creation of the pipeline in GUI 35

Set Attributes 36

Execution 38

Source Code 38

6. Results and Discussion 39

6.1. Feature tracking and visualization of consecutive timesteps 39

6.2. Feature tracking and selective visualization of consecutive timesteps . . 40

6.3. Feature tracking and visualization of non-consecutive timesteps 42

6.4. Feature tracking and enhanced visualization: change transparency of the objects 43

6.5. Execution time for Feature Tracking and Visualization modules 44

7. Conclusion and Future Work 47

Appendix A. Format of Input/Output/Intermediate Files 49

A.1. Input/Output files created during the execution of Feature Tracking module 49

A.2. Input/Intermediate/Output files created during the execution of Visualization module 55

Appendix B. Installing VisIt 57

References 60

List of Tables

6.1. Time to run Feature Tracking and Visualization for vorts and volmap dataset in VisIt on machine1 45

6.2. Time to run Feature Tracking for vorts and volmap dataset in VisIt on machine2 46

6.3. Time to run Feature Tracking for vorts and volmap dataset in AVS on machine2 46

7.1. Comparison of the Features of VisIt and AVS 47

List of Figures

1.1. Feature Tracking pipeline implemented in AVS [1] 3

3.1. Implementation of Feature Tracking Algorithm 9

3.2. Feature Extraction, image taken from [1] 10

3.3. Evolution of features over time [1] 12

3.4. Sort Node lists from two subsequent timesteps for Overlap Detection, image taken from [1] 13

3.5. Overlap Detection, image taken from [1] 13

3.6. Calculate Scoreboard to begin tracking of objects, image taken from [1] 15

3.7. Coloring scheme when two objects merge, image taken from [2] 17

4.1. VisIt components: Connectivity and Communication. Image taken from [3] 19

4.2. Description of the VisIt components [3] 20

4.3. Two possible pipelines in GUI to visualize data in VisIt 22

4.4. AVT pipeline of Source, Filters and Sink [4] 23

4.5. AVT pipeline created internally when executing Slice operator on a dataset. Image taken from [4] 24

4.6. Different cell types. Image from [5] 25

4.7. Different dataset types identified by VTK. In this thesis only vtkRectilinear dataset is handled. This image is taken from [6] 26

5.1. Steps to execute Object Segmentation and Feature Tracking module in VisIt 28

5.2. Initial windows that appear when VisIt starts 29

5.3. Pipeline created in GUI to perform Feature Tracking 30

5.4. Select single '.bov' file 30

5.5. Select multiple files. Click 'group': creates .visit file 30
5.6. Selection of a Plot 31
5.7. Operator Selection 31
5.8. Operator Attributes for FeatureTrack 32
5.9. Directory structure of the FeatureTrack operator source files 33
5.10. Steps to execute Visualization module in VisIt 34
5.11. Pipeline to visualize .poly files 35
5.12. Operator Attributes of the 'TrackPoly' operator 36
5.13. Plot Attributes of the 'PolyData' plot 37
6.1. Iso-surface visualization of 5 consecutive timesteps. Each object of the first timestep vorts1.data is color-coded randomly. vorts2.data, vorts3.data, vorts4.data, vorts5. data have their objects color-coded based on tracking information. 40
6.2. Selective Visualization: Part A - vorts1.data. is the first timestep, each feature is color-coded randomly. Part B - Color code of the three labeled objects is changed in the colormap file and visualization is performed again. 41
6.3. Selective iso-surface visualization of 5 consecutive timesteps. Colors of three objects in vorts1.data are modified as shown in Figure 6.2. vort2.data, vorts3.data, vorts4.data, vorts5.data have their objects colored as per tracking information 41
6.4. Iso-surface visualization of 3 non-consecutive timesteps. volmap3.bin is the first timestep hence each of its feature is color-coded randomly. volmap5.bin, volmap7.bin have their objects color coded as per tracking information 42
6.5. Iso-surface visualization of 3 consecutive timesteps. volmap3.bin is the first timestep hence each of its feature is color-coded randomly. volmap4.bin, volmap5.bin have their objects color-coded as per tracking information. 43

6.6. Enhanced iso-surface visualization: volmap3.bin is the first timestep hence each of its feature is color-coded randomly. Except the three labeled objects all objects have their transparency reduced. volmap4.bin, volmap5.bin have their objects color-coded as per tracking information. 44

6.7. Three nodes are picked and labeled. Information of each node is shown. The node value is used to change the color- code in colormap file 45

B.1. The Default directory structure when VisIt is installed 57

B.2. xmlEditor allows the user to give information to create a customized plugin 58

Chapter 1

Introduction

Scientific data visualization is the field of science which represents the usually complex and massive scientific data, through visual images. Such scientific data, often referred to as a 'dataset', can be the output of scientific simulation, output data of any laboratory experiments or sensor data in the fields of geology, medicine, physics, etc. These datasets can be of various sizes and structures. Visualization of these datasets is done using computer graphics to create visual images which can give an intuitive interpretation of the data, this can aid in better understanding of the numerical data. Scientific visualization is often a part of scientific process, where data visualization aids in observing desired features or region of interest, certain anomalies or evolving patterns. Customized computer graphics softwares, dedicated for scientific visualization are mostly used for this purpose.

In addition, time varying simulations are part of many scientific domains. Time varying datasets are outcome of experiments/simulations where a phenomenon is observed over a period of time and samples of data are collected at regular intervals. Such time varying simulations/observations are often used to study evolution of certain physical phenomenon. An example of this is visualizing cloud data to observe evolving cloud patterns over a period of time. Such time varying datasets can be difficult to comprehend by using standard visualization techniques [1]. Standard visualization techniques which generally employ surface/volume animation can fail to capture important features or evolving phenomenon, for example in the study of cloud formation over time, standard visualization techniques do not highlight continuously evolving cloud patterns effectively. Rather a technique is essential which can isolate region of interest

or features and highlight these phenomenon over time [7]. It is essential to have a region based approach like feature tracking and feature quantification. Also, the standard visualization tools available may not provide a quantitative description of the features or region of interest. For such purposes an automatic feature tracking tool is required which can highlight the stages in evolution of the desired features. This would essentially reduce the amount of data to process and also aid the scientists to focus on the region of interest by reducing the visual clutter. Nonetheless such a tracking tool needs to tackle three primary issues:

1. obtaining desired features and their quantitative information
2. tracking them over time
3. visualizing the features such that their evolution over time is easy to understand.

Vizlab at Rutgers has developed an efficient feature tracking application for scalar dataset in the AVS (Advanced Visualization System) environment [1, 7]. AVS is an interactive visualization software which allows the user to add plug-ins to perform specific visualization tasks [8]. The following steps describe the feature tracking process in AVS. Figure 1.1 shows the AVS pipeline for Feature Tracking. The major steps are:

- **Feature Extraction** - First identify and segment features of interest in the scalar dataset. Usually features are defined as threshold-connected components (See section 2.1.1 for details)

- **Feature Quantification** - After feature extraction each feature can be quantified. e.g.: Mass, Centroid, Surface Area etc. can be calculated.

- **Feature Tracking** - Next the evolution of the extracted features is followed over a period of time (See section 3.2 for details)

- **Visualization and Event querying** - Using the tracking information additional visualization steps like 'Event querying', which involves gathering information of evolution of features of interest, or even visualization of one object to view its evolution with time are performed [2].

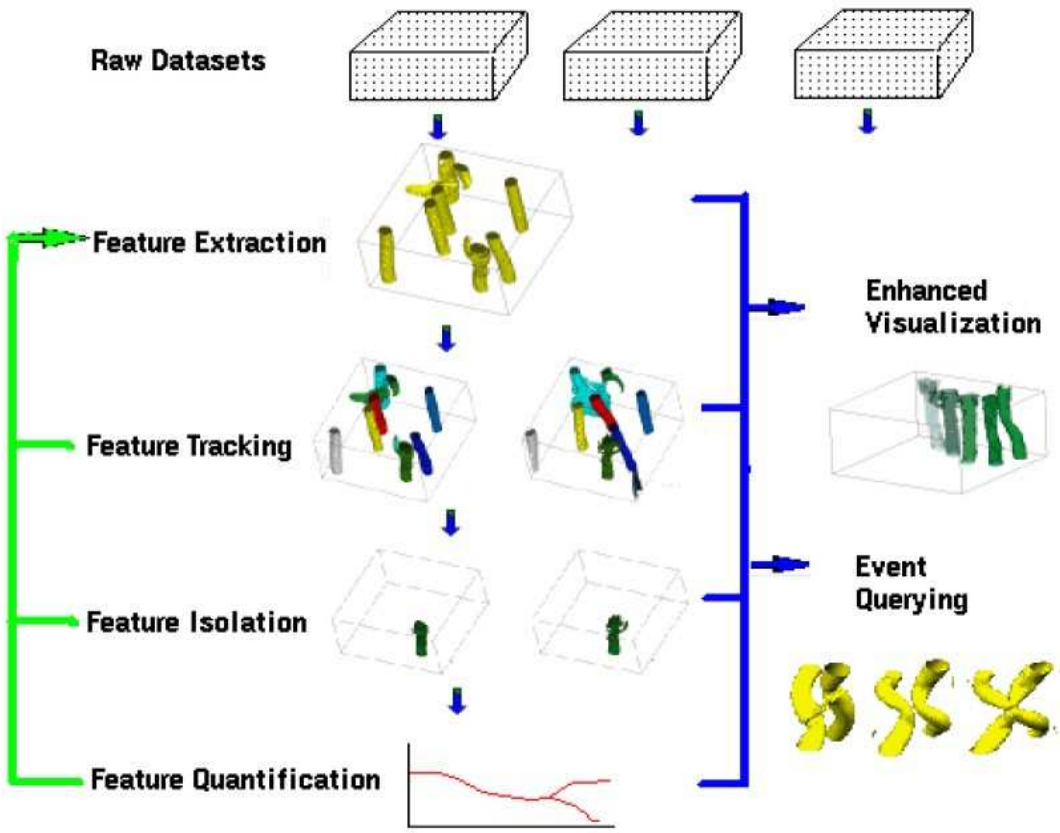


Figure 1.1: Feature Tracking pipeline implemented in AVS [1]

This Feature Tracking application has been successfully used by scientists from various labs. However, since AVS is a licensed software, the use of this application is restricted.

To make this application more useful it is essential to port the existing utility on an open source scientific visualization platform. This thesis presents the porting of the Feature Tracking utility developed earlier in AVS to VisIt [9]. VisIt, is a free interactive parallel visualization and graphical analysis tool for viewing scientific data. This thesis explores the VisIt environment for creating plug-ins to perform feature tracking and visualization. It also explains in brief about VTK (Visualization Tool Kit [10]), its format for data representation, which is inherently used in VisIt for all data modeling purposes.

The organization of this thesis is as follows: Chapter 2 is an overview of the Related Work done in this field. Chapter 3 gives the overview of the Feature Tracking algorithm as implemented in AVS. Chapter 4 gives an overview of the VisIt platform and the VTK datasets. Chapter 5 gives the implementation of Feature Tracking algorithm in VisIt. Chapter 6 details the results obtained based on the code developed and the various datasets tested. Finally Chapter 7 gives the limitation of the existing utility and proposes future work. Also, Appendix A give the formats of the input/intermidiate/output files created during Feature tracking and visualization. Appendix B gives the installation procedure for VisIt.

Chapter 2

Related Work

2.1 Importance of Feature Tracking in Visualization

Time varying simulations/observations are often used to study the evolution of certain physical phenomenon. This evolution information can help the scientists understand and analyze the inherent process or factors that cause the evolution. At times such information can help build predictive models. For example studying the cloud data to understand the cloud evolution can help scientists build a weather prediction model.

Standard visualization techniques which rely on surface and/or volume animation often fail to capture the evolving features and quantify their attributes. Scientists may have queries such as, what is the centroid of the biggest feature? What is the shape of the feature? What is its volume? In this regard, a feature tracking technique to automatically extract the features and quantify them [7] becomes very useful.

2.1.1 What are 'Features'?

In the context of Feature Tracking, 'Features' of the scalar datasets can be broadly defined as the connected coherent structures. The features can be obtained based on 'threshold' values, e.g., all the threshold connected components obtained using a region growing algorithm [7, 11]. The parameters to define features differs based on the application domain of the dataset.

2.2 Existing 3D scalar data Feature Tracking algorithms

Existing 3D scalar field feature tracking algorithms can be broadly classified into four categories which are briefly discussed below, please see [12] for a more in-depth discussion:

(a) **Attribute based feature tracking**:

An attribute-based feature tracking algorithm was proposed in [13, 14]. The algorithm extracts the features in each timestep and computes their attributes like size, location, etc. These attributes are then matched with features from subsequent time steps in a prediction-verification mode. Once the path of a set of feature's is found, a prediction is made by the authors using linear extrapolation with the path in the next timestep. The prediction is then compared with the real features in the next frame and a search for a match is performed. If one or more matches are found, they add the feature with the best match to the end of the path and continue into the next frame.

(b) **Higher dimensional iso-contouring based feature tracking**:

A higher dimensional iso-contouring feature tracking is explained in [15]. This method provides interactive tracking of user-selected local features. A local feature is defined as a connected component from an iso-surface or an interval volume computed from a scalar field. When tracking subsequent timesteps a 4D dataset is created from the two frames and 4D iso-contour is obtained. The 4D iso-contour is further sliced to obtain 3D iso-surface.

(c) **Surface propagation based feature tracking**:

In [1] iso-surface propagation is used and the seed set of the data to track the surfaces of the features. Surface propagation can identify merging, splitting, disappearance, continuum of the features in subsequent timesteps. To identify newly formed objects, a seed set is used.

(d) **Overlapping based feature tracking**:

In [16], five possibilities when tracking features in time varying datasets (continuation, creation, dissipation, bifurcation or amalgamation of features over subsequent timesteps) are defined. This classification is used in many volume tracking algorithms

[17, 7, 11]. In Chapter 3 this methodology is explained in detail. The current feature tracking algorithm in AVS is an overlapping-based feature tracking.

A feature is defined as a set of volume elements. These are extracted from the dataset as a best of nodes which compose the feature. Once extracted the set of features can be compared to the next timestep's set of features. This tracking algorithm works in two phases:

(1) VO-test: Overlap detection, which is to limit the candidates for best matching

(2) Best matching test: to find the correlation between features

Refer to section 3.2 for details of VO-test and Best matching test.

Use of Linked list Data Structure:

In [11], an overlapping-based feature tracking algorithm using a linked list data structure is described. Initially, features of a 3D scalar dataset are extracted generating an object list. Each object list contains information of object id, attributes and all nodes that are part of the object. After feature extraction of each timestep, all nodes of all the features are sorted according to the node ids, generating a sorted node list. Refer to section 3.2 for details.

Parallel and Distributed Feature Tracking:

The algorithm in [11] has limitation of handling large datasets due to limited processing capabilities of the single processors. [18] explains feature tracking of large datasets by using the powerful capabilities of parallel and distributed processing. In the distributed algorithm feature merge scheme, uses a binary swap algorithm [19] to communicate between processors. The tracking server sequentially operates on the output of the distributed feature extraction system and tracks features. A 'partial merge' strategy was also proposed. In this algorithm, after each processor does its own feature extraction, processors communicate with their immediate boundary neighbors to determine the local connectivity. The partial-merge data (given as a set of tables) is enough to reconstruct the full connectivity, which can be done by a visualization accumulator as a preprocess step to visualization (Explanation cited from [12]).

[2] has given the detailed explanation of implementing feature tracking in distributed mode. In this implementation each processor broadcasts the information of the local

minimum and local maximum data values from which a global minimum and maximum value of the data being processed in decided and used further for thresholding. [2] also mentions about implementing the feature tracking for large dataset on a single machine by simulating the behavior of multiple processors, such that subset of data processed by each processor in parallel is now handled sequentially by the single processor. This implementation was flawed as it would not find the global minimum or maximum for thresholding, instead it would just use the local minimum and maximum of the subset of data it was processing for the purpose of thresholding. This gave incorrect results. As a part of this thesis, the flaw was removed by first finding the global maximum and minimum value for thresholding and then using the same value for each subset of data handled by the single processor sequentially.

AMR (Adaptive Mesh Refinement) Feature Tracking:

In [20] a distributed feature tracking process for AMR datasets is described. AMR is used in computational simulations to concentrate grid points with varying resolutions [21]. Because features can span multiple refinement levels and multiple processors, tracking must be performed across time, across levels and across processors. The AMR tracking can be viewed as a grid of levels vs. time. Since some of the computation is redundant, tracking is computed temporally across the lowest refinement level and then computed across spatial levels of refinement. When a new feature is formed at a higher level of refinement it must be tracked in subsequent timesteps. The resulting visualization is represented as a 'Feature Tree'. (Explanation cited from [12])

In this thesis we build on the prior work done in Vizlab based on Feature Tracking explained in section 2.2. In this thesis the feature tracking application is ported to VisIt [9] and visualization module is separated from feature tracking module to provide selective visualization.

Chapter 3

Overview of the Feature Tracking Algorithm Implementation

The feature tracking algorithm as implemented in the AVS (or VisIt) environment, can be broadly divided into 3 parts (See Figure 3.1):

1. Feature extraction and quantification
2. Feature Tracking
3. Visualization of the Features

The current algorithm implementation handles only 3D structured dataset (rectilinear

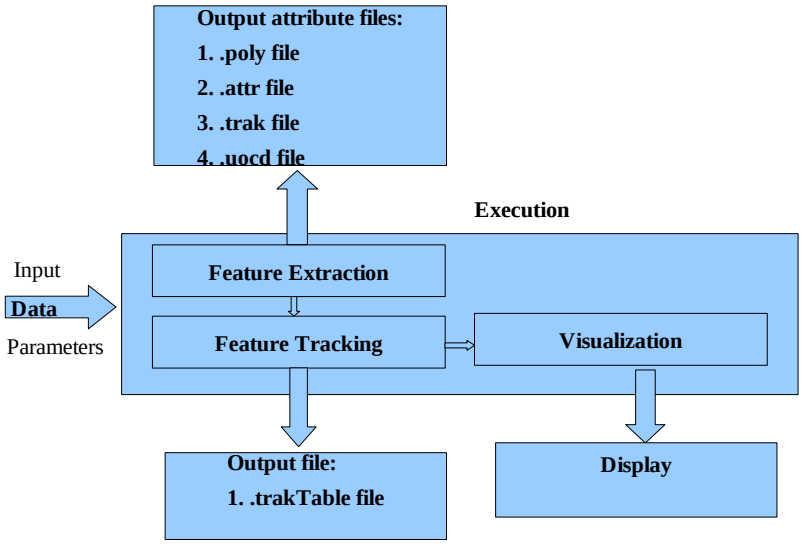


Figure 3.1: Implementation of Feature Tracking Algorithm

dataset type, refer section 4.5.2 for details).

3.1 Feature Extraction and Quantification

The most important part of feature tracking is defining what features to track. General methods like segmentation [22], volume intervals define features based on some connectivity criteria. In the current implementation, the features are defined as connected voxels, satisfying the threshold criteria [7]. These components are later visualized using iso-surface routine. Following is the procedure to extract features:

1. **Process Input**:

a. Read the point data, point co-ordinates, cell connectivity of the dataset (field data in AVS or vtkdataset in VisIt).

b. Mark all the points above user defined threshold. For all these points create a list of cells incident on each point.

2. **Object Segmentation** (Figure 3.2 shows the Feature extraction and Object Segmentation process):

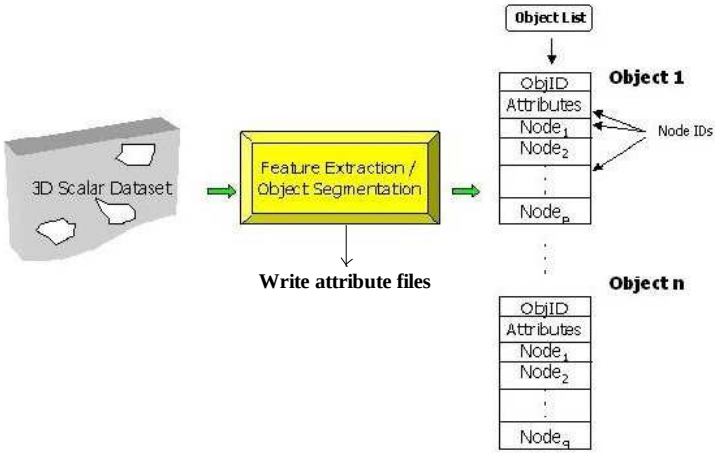


Figure 3.2: Feature Extraction, image taken from [1]

Segmentation works in a loop until all possible features are extracted. In each loop start with a 'seed' defined as the highest unchecked data value above the threshold.

- Create a list of all the cells incident on each node with this data value. Mark the node as 'used'. Create a new object . Assign this object a unique *object number*.
- Based on the connectivity criteria [23] each cell is tested for inclusion in the

object. Add all the cells which pass the inclusion test to the object's cell list and assigned each cell the object's number (if the cell is not already assigned any object number).

- If no cell in the object's cell list has a previously assigned number, a new feature is found. Increment the object count.

- If any of the cell is already assigned an object number implies that the current object is connected to another object. Merge the current object's cell list with the previous object and assign the previous object's unique number to all the cells in current objects list. Delete the current object.

3. **Feature Quantification**:
Once the features are extracted, their attributes like its centroid, volume, mass, moment are calculated and stored in files for each timestep.

3.2 Feature Tracking

Once the features are extracted, they are characterized by the following evolution of *Continuation, Bifurcation, Amalgamation, Dissipation or Creation*[1][7]. These events are illustrated in the Figure 3.3

Continuation: Rotation or translation of the feature may occur and it may grow in size or become smaller in the next timestep

Bifurcation: A feature separates into two or more features in the next time step

Amalgamation: Two or more features merge in the next timestep

Dissipation: A feature becomes weak and fades into the background i.e., it falls below the threshold value

Creation: A new feature appears in the next timestep (it cannot be matched to any feature in the previous timestep)

Matching features from one timestep to another is known as the *correspondence problem*. [7] explains in detail the correspondence problem and the matching test for

[1]The definitions are taken from [7]

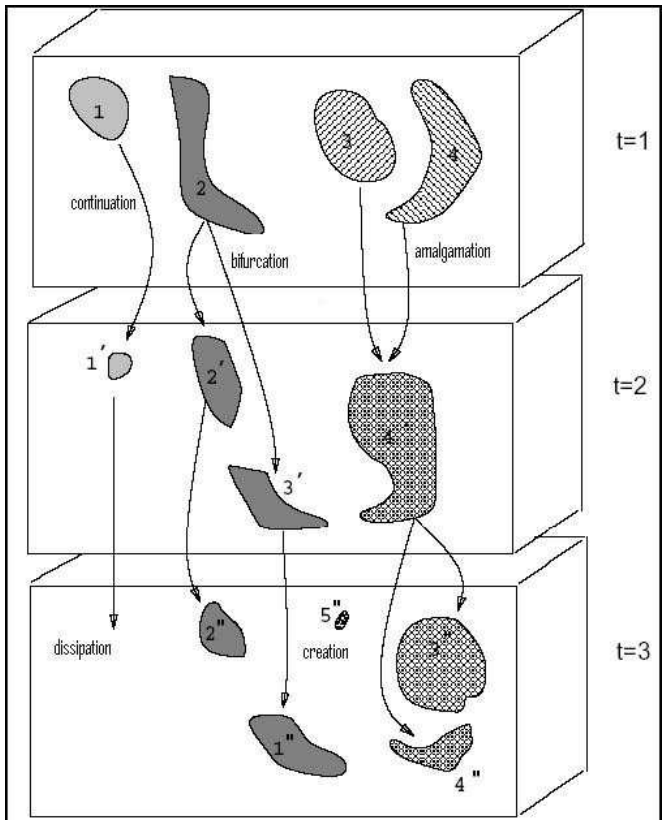


Figure 3.3: Evolution of features over time [1]

overlapping based volume (set of voxels) tracking, using octree data structure. Following is the brief description of the overlapping based volume tracking [7] as implemented. This tracking algorithm works in two phases:

(1) **VO-test**: Overlap detection, limits the candidates for the best matching test. For each timestep create a list of all the nodes in each object and sort the list (See Figure 3.4). Compare the sorted list of timestep t_i and t_{i+1} to detect overlap and store the result in the overlap table.

Implementation of Overlap Detection test:

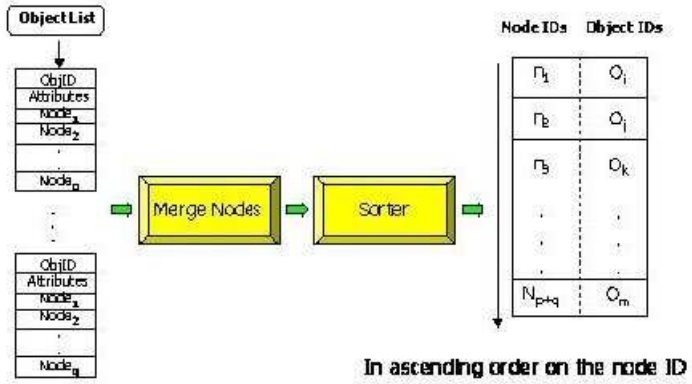


Figure 3.4: Sort Node lists from two subsequent timesteps for Overlap Detection, image taken from [1]

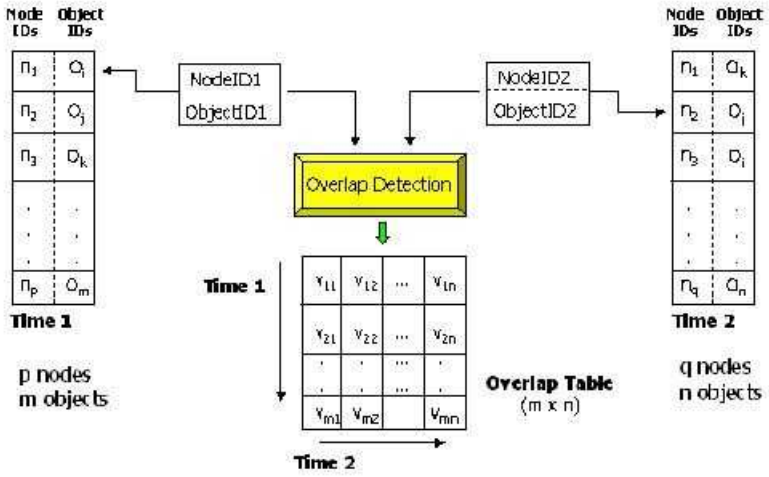


Figure 3.5: Overlap Detection, image taken from [1]

1. Create an Overlap table (2D array of size m*n where m and n are number of objects in timesteps t_i and t_{i+1} respectively)

2. Start from the first node in each of the sorted list for timestep i and subsequent timestep i+1 respectively

3. Compare the values

4. If the values of the nodes match, update the overlap table corresponding to the object no. to which each of the node belongs in timestep i and i+1.

5. Else if value of node in list 1 is > list 2 go to the next node in list 2 until the two values match or the list 2 reaches its end

6. Else go to the next node in list 1 till the two values match or the list 1 is over.

After the overlap table is computed, it is used to do best matching. The best matching test checks all combinations of overlap to determine whether an object continues in the next timestep, or breaks up in two or more objects in the next timestep.

Figure 3.5 depicts the overlap detection process.

pseudo-code of overlap detection (taken from [1]):

p_i /*pointer to nodes of timestep i */

p_{i+1} /*pointer to nodes of timestep i+1 */

while($p_i <$ NumNodes1 or $p_{i+1} <$ NumNodes2)

$R_1 = p_i; R_2 = p_{i+1};$

if $R_1.NodeID == R_2.NodeID$

then $Overlap_table[R_1.NodeID, R_2.NodeID]$

p_i++

p_{i+1}++

else if $R1.NodeID > R2.NodeId$

then p_{i+1}++

else

p_i++

(2) **Best matching test**: Finds the correlation between the features (explanation taken from [1]).

The correspondence metric to find the best match is given as:

$$M = O_A^i * -O_B^{i+1}$$

(O_A^i refers to a feature A in timestep i and

O_B^i refers to a feature B in timestep $i+1$)

$$M = max(O_A^i - O_B^{i+1}, O_B^{i+1} - O_A^i\ )$$

The best match is the one minimizing M. If two objects exactly match, then M = 0.

Alternatively, the correspondence metric can be approximated as

$$M = O_A^i \bigcap O_B^{i+1}$$

This is maximized when two objects are identical. To normalize the result of matching, R can be computed as below:

$$R = \frac{Volume(O_A^i \bigcap O_B^{i+1})}{\sqrt{Volume(O_A^i)Volume(O_B^{i+1})}}$$

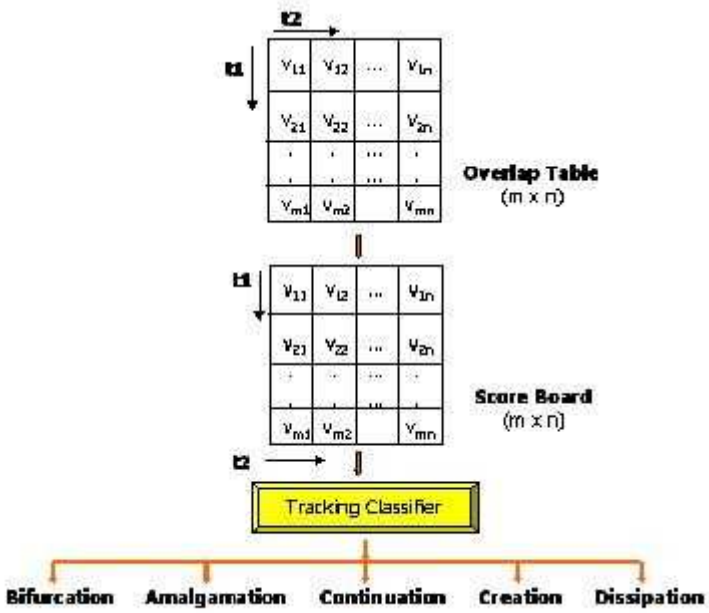


Figure 3.6: Calculate Scoreboard to begin tracking of objects, image taken from [1]

Figure 3.6 depicts the scoreboard calculation to begin classification of features. Thus to summarize, Feature Tracking algorithm works as follows, for the two consecutive timesteps t_i and t_{i+1} [1]:

1. Extract the features from the two datasets and store each in a sorted link list.

2. Construct a forest which is nothing but union of all the features O_A^i (The O_A^i refers to a feature A in timestep i) in given timestep

$$F_i = \bigcup_{p \in t_i} O_p^i$$
$$F_{i+1} = \bigcup_{q \in t_{i+1}} O_q^{i+1}$$

*/*Use O_p^i as template for matching */*

3. For each feāture $O_p^i \in F_i$ merge it into the forest,F_{i+1}, Identify all the overlapping regions of O_p^i in t_{i+1}. Store this in a list called OverlapO_p^i[ ]

*/*Determine bifurcation and continuation*/*

4. For all combinations of features in OverlapO_p^i[ ]

5. Compute O_p^i*-UOverlapO_p^i[]

If the lowest difference is below the tolerance,

Mark O_p^i as bifurcating into the object and remove them all from the search space

Next O_p^i

*/*Else, Determine Amalgamation*/*

6. For each remaining feature in O_q^{i+1} merge it into the forest, F_i and test for amalgamation

*/*This is the same as bifurcation with the inputs*/*

7. Take the remaining O_p^i in t_i as dissipation

8. Take the remaining O_q^{i+1} in t_{i+1} as creation

3.3 Visualization

Once the features are extracted and tracked visualization is performed using iso-surface information (in AVS, visualization is performed for each time step after tracking and in VisIt visualization is implemented as a separate module which runs after feature tracking is completed). Each object in the first timestep has a color-code (R,G,B value) assigned to it (In AVS this color-code is based on volume, mass or randomly based on the user option, in VisIt user can either write own colormap file or VisIt assigns random colors). A color table is internally created based on the color-code for each object. In the subsequent steps, objects are assigned colors based on tracking

information.

When an object bifurcates in next timestep each child object has the same color as

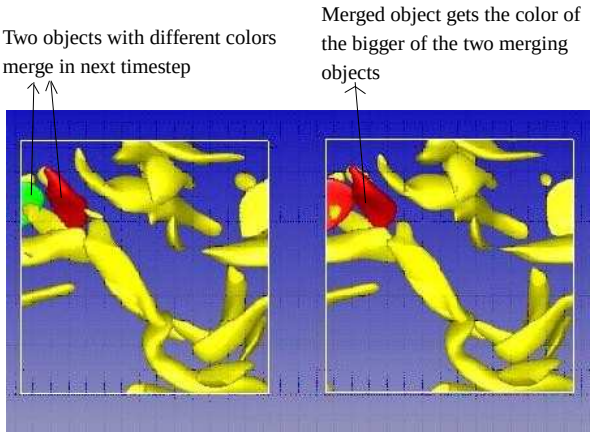


Figure 3.7: Coloring scheme when two objects merge, image taken from [2]

that of the parent object. When two or more objects merge, the new object gets the color of the bigger object. Figure 3.7 shows color-coding scheme when two objects merge in subsequent timesteps. If the object is created in a subsequent timestep, then it is assigned a color based on the original coloring scheme selected.

Chapter 4

VisIt

4.1 What is VisIt?

Scientific data visualization is an important part of many scientific experiments (see section 1 for more details). With more and more powerful machines now available for scientific computations, the datasets can be massive. With advancements in Computer Graphics, the emphasis on realistic rendering of scientific data has increased. With the efforts of many developers and research organizations, many open source scientific data visualization tools are now available which can handle large datasets and provide flexible functional modules for effective rendering. Some of these tools cater to specific datasets like GGobi is an open source visualization tool for exploring high dimensional data [24]. Another example is Paraview, which is an open source parallel visualization application to efficiently handle large datasets on the distributed systems [25].

This thesis explains the feature tracking application developed in the VisIt environment. VisIt is a free, open source, interactive, parallel visualization and graphical analysis tool for viewing scientific data on the Unix and PC platforms. It has a rich set of functionalities to handle scalar, vector and tensor field visualization. VisIt also provides qualitative and quantitative analysis of rendered data. It is a platform which supports multiple mesh types and is extensible with provision of dynamically loaded plug-ins. It can handle large datasets and has parallel and distributed architecture for visualization. In addition to these key features, VisIt is supported on multiple platforms and supports C++, Python and Java interfaces which enables a user to provide an alternate user interface or plug-in user created modules to VisIt (these key features are described in [26]).

4.2 VisIt Architecture

VisIt is composed of multiple, separate processes, which are sometimes referred to as components. Figure 4.1 shows the interaction of various VisIt components. Table in Figure 4.2 gives a brief description of the VisIt components[1].

At the lowest level, these separate component programs communicate over sockets.

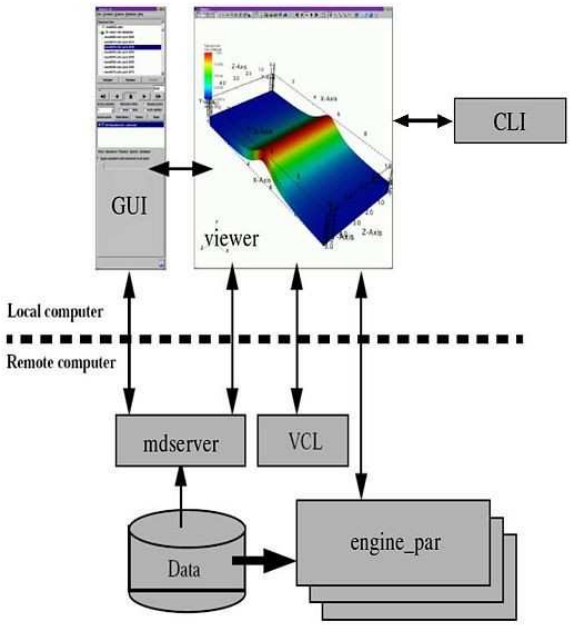


Figure 4.1: VisIt components: Connectivity and Communication. Image taken from [3]

Since each of these components must communicate, a common protocol is needed to simplify the passage of messages over the socket. VisIt uses state objects, which are essentially C++ structs. State objects are sent over the socket in between VisIt components and on either side, appropriate action is taken when they are received. Each state object knows how to serialize itself onto VisIt's sockets and can be reconstituted on the other side of the communication once the entire state object has been read from the socket. Two separate layers are built over the top of the sockets:

1. The first is for exporting state. The viewer keeps all of its state in various instances of VisIt's AttributeSubject class. UI modules subscribe to this state. Thus, when state

[1]the table is taken from [3]: The component-ized design of VisIt

Name	Overview	Dependencies
viewer	1. centralizes all of VisIt's state. When the state changes, it notifies the other components of the state changes. 2.he viewer is responsible for managing visualization windows, which often includes doing rendering in those windows.	1. VTK (for data models and plotting), 2. Qt (for popup menus), 3. OpenGL (for plotting)
gui	Provides a graphical user interface to control VisIt.	Qt
cli	Provides a command line user interface to control VisIt.	Python
vcl	Launches jobs on remote machines. The VCL sits idle on remote machines, communicating with the viewer and waiting for requests for jobs to launch. When these jobs come up, it launches them. The purpose of this module is to spare the user from having to issue passwords multiple times.	None
mdserver	The mdserver browses remote file systems, meaning it produces listings of the contents of directories. It also opens files (in a lightweight way) to get meta-data about a file, allowing the user to set up plots and operators without an engine.	1. Many I/O libraries (Silo, HDF5, HDF4, Exodus, NetCDF, more) 2. VTK
engine	The engine performs data processing in response to requests from the viewer. There are both parallel and serial forms of the engine (called engine_ser and engine_par respectively). Rendering is sometimes performed by the engine, although it is also performed on the viewer.	1. Many I/O libraries (Silo, HDF5, HDF4, Exodus, NetCDF, more) 2. VTK 3. Mesa (actually mangled Mesa) and/or OpenGL

Figure 4.2: Description of the VisIt components [3]

changes on the viewer, the AttributeSubjects automatically push this state out to its subscribers.

2. The second is for remote procedure calls (RPCs). When a component wants another component to perform an action, it issues an RPC. For example Viewer causing the mdserver to perform an action, such as opening a file or Viewer causing the engine to perform an action, such as drawing a plot. [State objects as defined in [27]]

4.3 Visualizing 3D Dataset in VisIt

VisIt has three kinds of plug-in to read a dataset, manipulate and visualize it. To visualize a 3D dataset in VisIt, a pipeline consisting of a file reader, operator and plot plug-ins can be created (in the GUI). Following is the brief description of each of these plug-ins:

1. **Database Plug-in**:

This plug-in is loaded when the user selects a dataset file to read/open (mdserver shown in the Figure 4.1 loads this plug-in). Depending on the file extension a database file reader plug-in is internally loaded dynamically to read the file correctly. If VisIt does not have a built-in reader for the data type, a user can write a new database file reader plug-in. The dataset reader outputs the data in generalized vtkdataset (see section 4.5 for details of vtkdataset) format. Output of the plug-in is generally in the vtkDataset format.

2. **Operator Plug-in**:

Once the data is read by the database reader plug-in, the data is now available to be visualized. The input as well as output data to the plug-in is generally in the vtkDataset format (see section 4.5 for details of vtkdataset format). Often, one wants to manipulate the data or apply a filter to make the data visualization more meaningful. VisIt provides it own set of operators such as iso-surface, iso-volume, slicing etc. Users can write their own operator plug-in to perform specific filtering operations on the data. User should select an operator plug-in after a data file is opened and a plot is selected (When the engine shown in Figure 4.1 executes, it loads the selected operator). The operator is dynamically loaded. Multiple operators can be sequentially applied.

3. **Plot Plug-in**:

Plot plug-in is used to visualize the data. Again, VisIt provides a rich set of plot plug-ins like pseudocolor, mesh, histogram etc. User can write a plot plug-in to perform special visualization tasks. User should select a plot plug-in after opening a data file. The plot plug-in is dynamically loaded. Input to the plug-in is generally in the vtkDataset format (see section 4.5 for details of vtkdataset).

Figure 4.3 shows the two possible pipelines a user can create in GUI to visualize data in VisIt. Besides the above mentioned plug-ins, when a user selects a file and

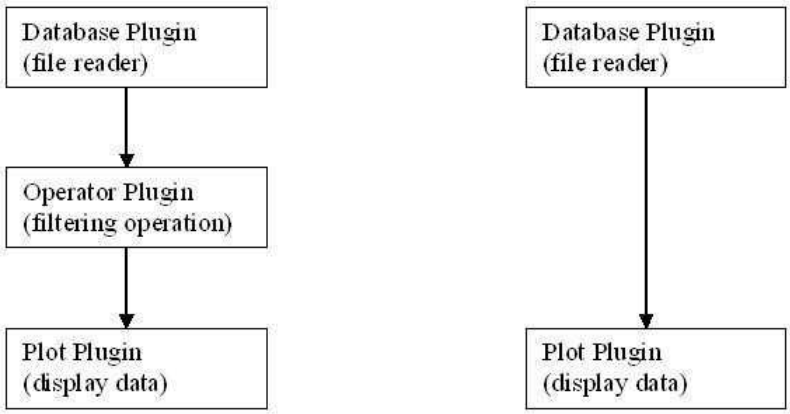


Figure 4.3: Two possible pipelines in GUI to visualize data in VisIt

applies plot and/or operator, VisIt internally creates its own pipeline such that user selected operators and plots are correctly loaded and executed.

4.4 AVT in VisIt

AVT stands for the 'Analysis and Visualization Toolkit'. It is basically a data flow network design created within VisIt. Its base types are data objects and components. It provides the network for data processing based on the user request through the GUI.

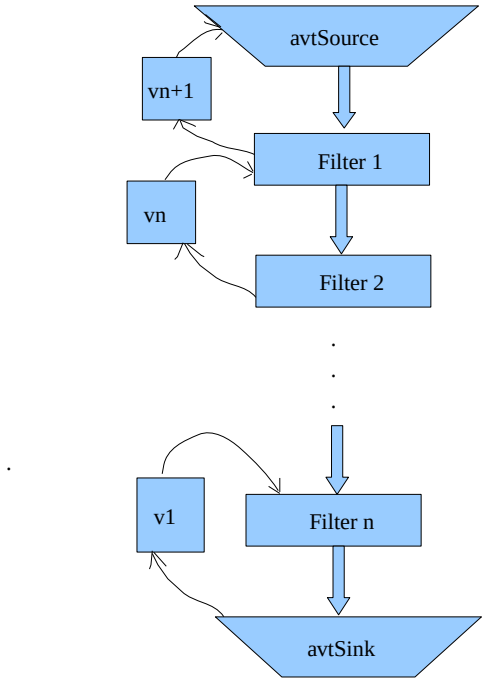


Figure 4.4: AVT pipeline of Source, Filters and Sink [4]

4.4.1 Overview

The components of AVT can be sources, sinks, or filters. 'Sources' only output data objects, 'sinks' only input data objects, and filters have both an input and an output. A pipeline in AVT data flow network has a source (example file reader) followed by one or many filters followed by a sink (example a rendering engine). Pipeline operation is based on demand. Generally the sink starts a 'pull' which forces the sink to generate an update request that propagates up the pipeline through each filter in the pipeline and terminates on the source. Once the source is reached, the source outputs the requested data, and then execute phases propagate reverse down the pipeline until the sink is reached. The network uses 'Contracts' to propagate requests. See Figure 4.4. (Description taken from [4]).

4.4.2 Use of AVT in VisIt pipelines

The engine assembles AVT network and executes it. The Viewer and mdserver also use AVT libraries (which are basically C++ abstract or concrete classes). When a

user opens a file the mdserver opens an avtFileFormat and reads the data. When the user selects a plot or an operator the engine creates an AVT network to execute the call. When 'Draw' button is clicked, AVT network is executed. The 'FeatureTrack' operator plug-in created for feature tracking uses the 'avtTimeLoopFilter' from the AVT library to keep track of the current timestep. Figure 4.5 shows the AVT pipeline created internally in VisIt when the user selects the Slice operator to display slice of input dataset (Description from [4]).

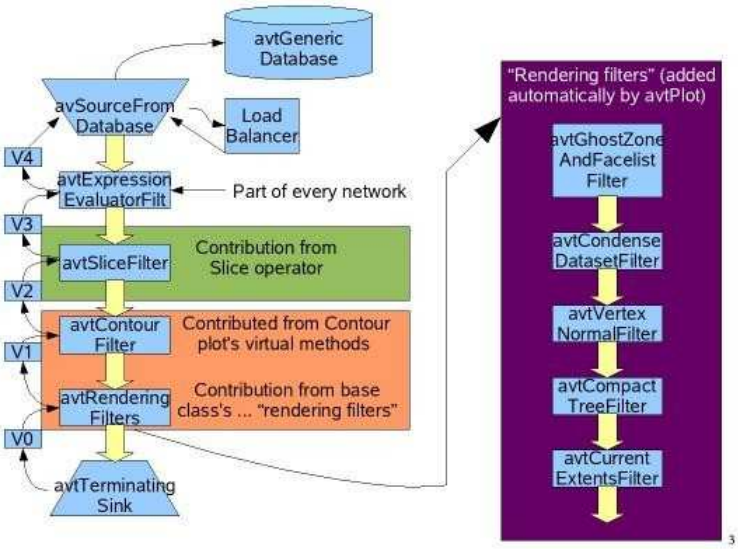


Figure 4.5: AVT pipeline created internally when executing Slice operator on a dataset. Image taken from [4]

4.5 vtkDataset

VisIt uses the generalized 'vtkdataset' to represent scientific datasets with different structures and attributes and for data modeling. The vtkDataset is defined in VTK (Visualization Tool Kit - an open source Image Processing and Visualization tool [10]). Once the data is read into the vtkdataset form, VisIt can use different algorithms for data manipulation and rendering. Following subsections give a brief description of the structure and topology of the vtkdataset[2].

[2]all VTK definitions taken from [28]

4.5.1 vtkDataset structural description

- **Structure**: Dataset structure has two parts: *topology* and *geometry*. Topology is the set of properties invariant under certain geometric transformations like rotation, scaling etc. Geometry is instantiation of the topology, the specification of position in 3D space. The structure consists of cells and points.

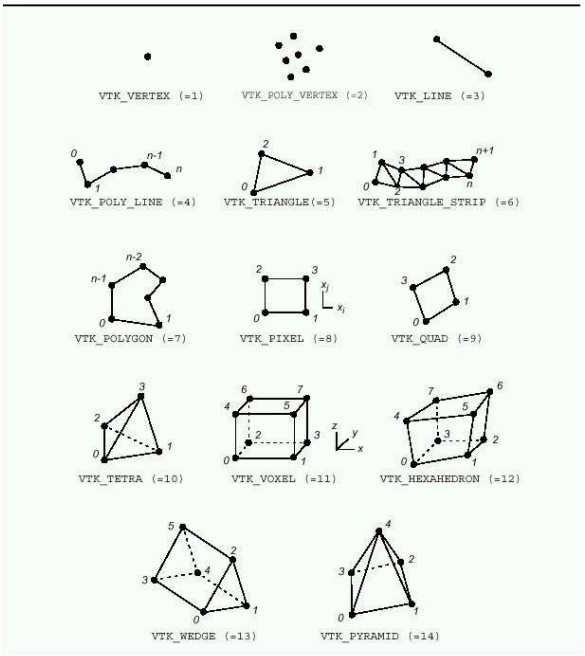


Figure 4.6: Different cell types. Image from [5]

  - **Points**: Points are the discrete locations where data is known. Points specify the geometry of the dataset.
  - **Cells**: Cells are the fundamental building blocks of visualization system. They define the topology of the dataset. Each cell is an ordered list of points called connectivity list. The cells can be 0,1,2 or 3D. Examples of cell types are vertex, polyvertex, line, triangle, quadrilateral, polygon, voxel, hexahedron etc. Figure 4.6 shows different cell types identified by VTK.

- **Attribute Data**: Dataset attributes are additional information associated with

the geometry and/or topology. Example of attributes are scalars, vectors (data with magnitude and direction), normals (direction vectors), texture co-ordinates etc.

4.5.2 Types of vtkDataset

VTK defines following types of datasets, characterized by structure whether its regular (single mathematical relation within the composing cells and points) or irregular. Figure 4.7 shows different dataset types identified by VTK.

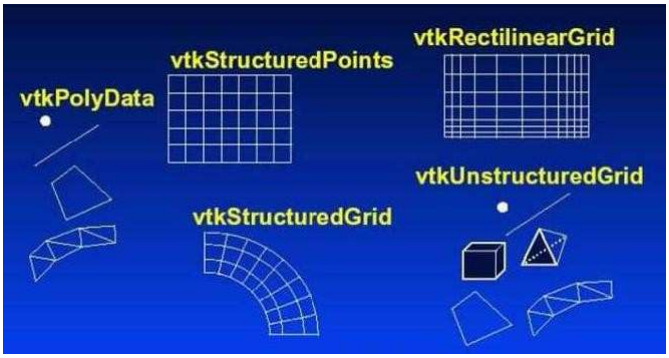


Figure 4.7: Different dataset types identified by VTK. In this thesis only vtkRectilinear dataset is handled. This image is taken from [6]

- **Polygonal Data**: Topology and geometry are unstructured, cells composing the dataset vary in dimension (0,1 and 2D).
- **Structured Points**: Topology and geometry are structured and can be implicitly represented. Also called uniform grid. Points and cells are arranged on regular rectangular lattice.
- **Rectilinear grid**: Topology is regular and geometry is partially regular. Points and cells are arranged on regular rectangular lattice. Topology is implicitly represented while geometry is represented by specifying x,y z co-ordinates.
- **Structured grid**: Topology is regular and geometry is irregular. Composing cells are quadrilaterals or hexahedron.

- **Unstructured Points**: No topology and unstructured geometry. Points are irregularly located in space.
- **Unstructured Grid**: Most general form of dataset. Topology and geometry are both unstructured.

Chapter 5

Feature Tracking and Visualization of 3D Scalar Datasets in VisIt Environment

The Feature tracking implementation is divided in two major algorithmic modules as follows:

1. Object Segmentation and Feature Tracking
2. Visualization using iso-surface information

5.1 Object Segmentation and Feature Tracking

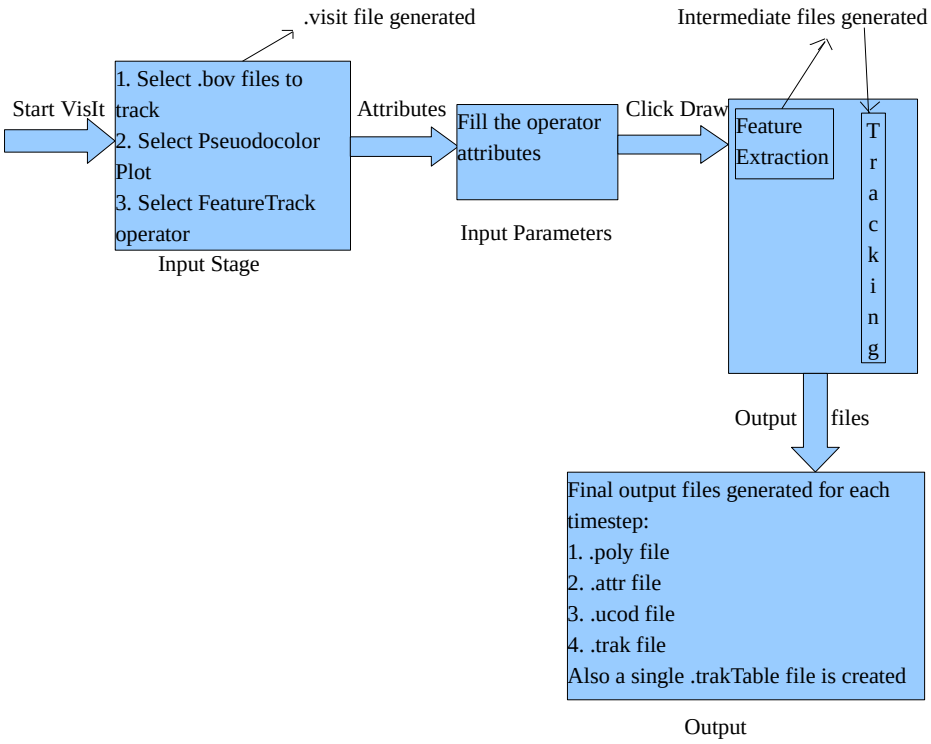


Figure 5.1: Steps to execute Object Segmentation and Feature Tracking module in VisIt

For each selected timestep object segmentation is performed based on the threshold given by the user followed by tracking of features. For each timestep attribute information of each object is stored in files. Also the tracking information of all the timesteps is stored in a single file. Figure 5.1 gives the description of the actual process to execute Feature Tracking module in VisIt. In the following subsection, each stage of execution is explained.

5.1.1 Using the Software

Creation of the pipeline in GUI

Figure 5.2[1] shows the initial window when VisIt starts.

As shown in Figure 5.3 a pipeline of '.bov[2] file' reader, 'FeatureTrack' operator to

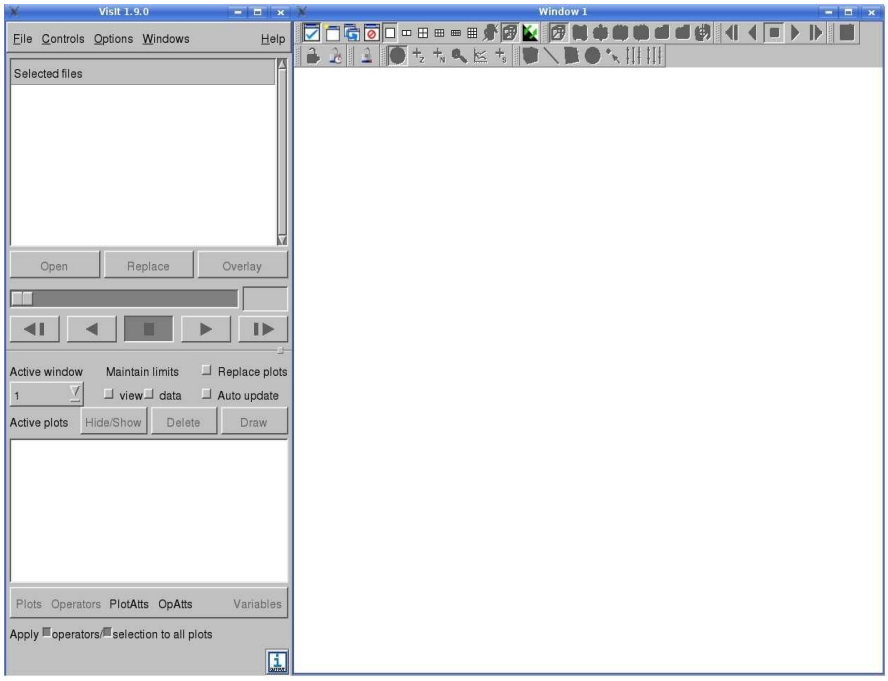


Figure 5.2: Initial windows that appear when VisIt starts

[1]Plot and operator selection is disabled. Plot selection is enabled when file is opened and operator selection is enabled when file is opened and plot is selected

[2].bov is a file reader built-in in VisIt. '.bov' files are used to read binary datasets in VisIt. Its format is given in the Appendix A. It has additional information to read in binary data

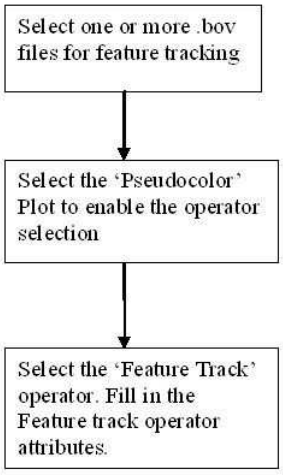


Figure 5.3: Pipeline created in GUI to perform Feature Tracking

perform object segmentation and feature tracking and 'Pseudocolor' plot is to be created to perform feature tracking. The plot selection is actually not required in this part as visualization is a separate process, but since VisIt does not permit selection of any operator without selection of a plot, plot is selected.

VisIt allows selection of single or multiple files at a time. If multiple files are selected such that the operator or plot is applied to each file, then VisIt gives an option of grouping the files (see Figure 5.5) to create a .visit file. The file reader then guesses the cycle number for each file in the group. Figure 5.4 and Figure 5.5 show how single and multiple files respectively can be selected.

After the file is selected, it should be opened (Figure 5.2 shows 'open' button, select a

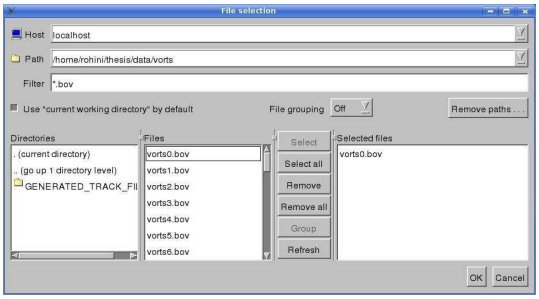


Figure 5.4: Select single '.bov' file

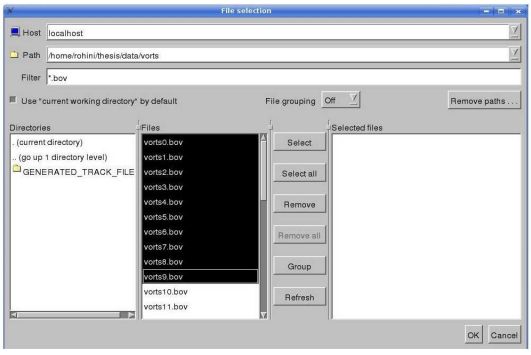


Figure 5.5: Select multiple files. Click 'group': creates .visit file

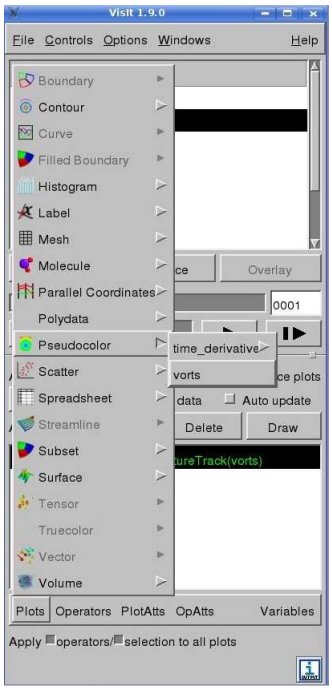


Figure 5.6: Selection of a Plot

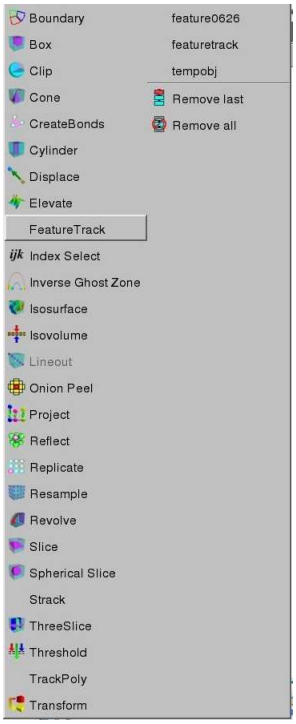


Figure 5.7: Operator Selection

file and click on 'open') to enable selection of plot and operator. Figure 5.6 and Figure 5.7 show how 'Pseudocolor'plot and 'FeatureTrack'operator are selected.

Set attributes

Most of the plots and operators have multiple attributes, which can be set as per specific tasks to perform. Figure 5.8 shows the attributes that should be set for 'FeatureTrack' attribute. Click on the 'opAtts' on GUI and select 'FeatureTrack' operator to set its following attributes:

a. **inputBaseFilePath**: The user needs to specify the complete path of the files (.bov files) to be tracked. Actually the user specifies the 'labelname'. e.g.: For the vorticity dataset, the files are named vorts1.data, vorts2.data, and so on. Hence the labelname to be entered is vorts.

b. **initialTimeStep, FinalTimeStep, timestepIncrement**: Enter the number of the timestep from where tracking should start, end and also the increment in which files should be tracked e.g.: For the vorticity dataset, if the start value, end value and

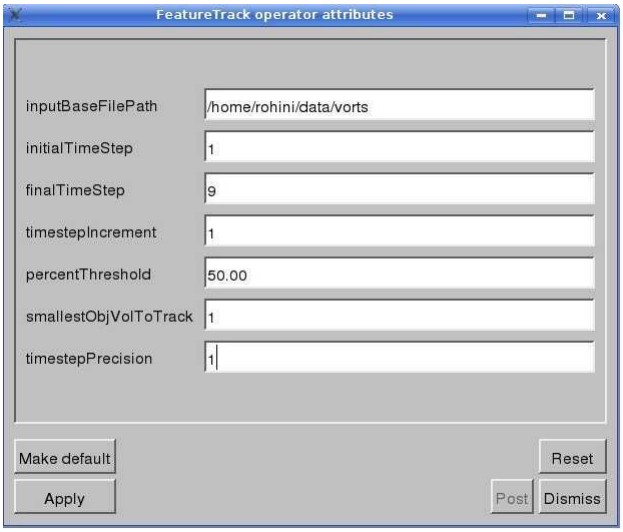


Figure 5.8: Operator Attributes for FeatureTrack

increment are 1,5 and 2 respectively then the files which are going to be tracked are vorts1,vorts3,vorts5.

c. **percentThreshold**: The actual threshold calculated for segmentation is based on this input.

$thresh = (max\_data\_value - min\_data\_value) * percentThreshold/100$

d. **smallestObjVolToTrack**: While tracking datasets large number of small objects which are extraneous to the regions of interest, the user can choose to blank out these unwanted features smaller than a certain volume by using this slider. (Volume of an object is defined by the number of nodes contained within it). (Explanation from [2]).

e. **timestepPrecision**: This indicates the style of numbering of the files to the software. e.g.: If the files to be tracked are named vorts01,vorts02, etc, the user still enters 1 and 2 as the start and end value and chooses a precision of 2. This is done in order to keep the values passed as start and end values uniform without any trailing zeroes. The default precision is 1 (explanation from [2]).

Execution

After selecting the plot and operator and setting the operator attributes, click on the 'Draw' button to begin feature tracking. When feature tracking is successfully performed, for each timestep attributes such as moments, centroid, mass etc are calculated and attribute files are written. Following attribute files are written:

- .trak file
- .attr file
- .uocd file
- .poly file[3]

Apart from these file one .trakTable file is created which is used to track features across timesteps. Appendix A has the description of all these files. These files are created in the subdirectory under the data directory from where the .bov files are selected.
$< pathof.bovfiles > /GENERATED\_TRACK\_FILES/ < attributefiles >$

Source Code

The source code for this part is in the 'FeatureTrack' operator directory.
$< visitpath > /src/operators/FeatureTrack/$
When the operator plug-in is created using XMLeditor, VisIt creates certain files such

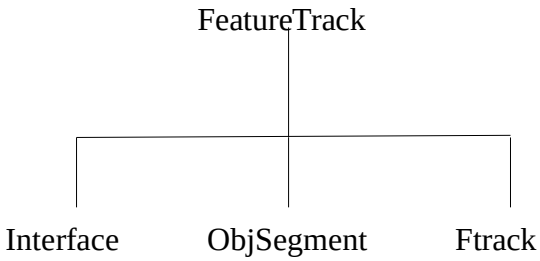


Figure 5.9: Directory structure of the FeatureTrack operator source files

that the plug-in is correctly loaded when user selects it. Also the attribute (.C and

[3]this file has the co-ordinate and connectivity information to visualize each object of given timestep

.h) files parse the attributes set by the user for the operator. The most important files created by VisIt are the $avt < operatorname > Filter.C$ and $avt < operatorname > Filter.h$ files. User can add the code to implement desired functionality in these files. Apart from files created by VisIt, following folders/files are present in the 'FeatureTrack' operator source code (see Figure 5.9):

Interface: This folder contains files to validate data given by user through the user interface.

ObjSegment: This folder contains files which perform object segmentation.

Ftrack: This folder contains files that perform feature tracking.

See Appendix B for details of creating a new plug-in and using the XMLeditor.

5.2 Visualization Using the Iso-surface Information

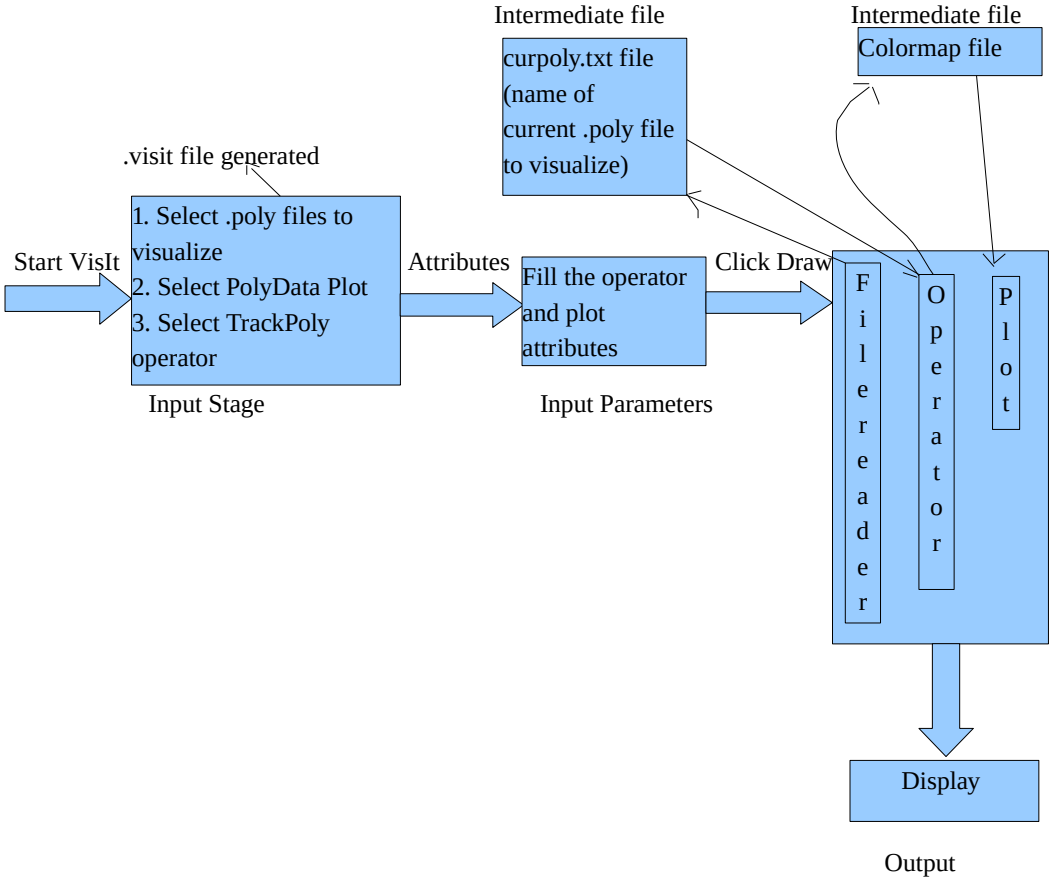


Figure 5.10: Steps to execute Visualization module in VisIt

In the second part, visualization is performed. The advantage of separating the feature tracking process from the visualization is that the user can now select to visualize all or subset of timesteps tracked in the first part. Sometimes interesting features begin to evolve a few timesteps from the first frame tracked. In such cases, its important that the user has the flexibility to select the timestep to start visualization. Also, with the separation of visualization, the user now has the flexibility of assigning specific colors to certain objects of interest in first timestep selected for visualization. User can keep intact the visual tracking of other objects or fade others so that evolution of features of interest only is tracked over the remaining timesteps. A colormap file is created for each timestep which is used for coloring each object in that timestep. Figure 5.10 gives the description of the actual process to execute Visualization module in VisIt. In the following subsection, each stage of execution is explained.

5.2.1 Using the Software

Creation of the pipeline in GUI

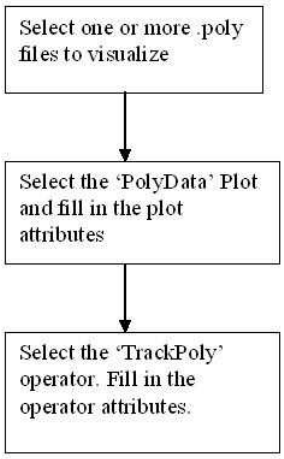


Figure 5.11: Pipeline to visualize .poly files

Figure 5.11 shows how pipeline of plug-ins is created to perform visualization of .poly files. A file reader is written so that .poly files can be read into VisIt for visualization. Operator 'TrackPoly' is a plug-in written to create colormap (if first timestep

is visualized) or update the colormap with the aid of the track table (for subsequent timesteps). 'PolyData' plot is written to visualize all the objects in a given timestep. It reads the colormap written by the 'TrackPoly' operator to visualize the objects. This plot is customized such that colormap created by 'TrackPoly' can be used to map colors to the objects.

Set Attributes

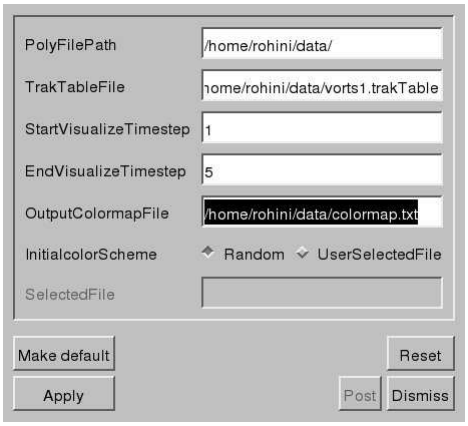


Figure 5.12: Operator Attributes of the 'TrackPoly' operator

Figure 5.12 shows the attributes that should be set for the 'TrackPoly' operator and Figure 5.13 shows the attributes that should be filled for the 'PolyData' plot. Click on the 'opAtts' button in the GUI and select the 'TrackPoly' operator to set the following attributes:

a. **polyFilePath**: Give the path of the folder where the .poly files reside.

b. **TrakTableFile**: Give the name and complete path of the .trakTable file generated in the feature tracking module.

c. **StartVisualizeTimestep, EndVisualizeTimestep**: Enter the number of the timestep from where visual tracking should start and end. The operator will start reading from the track table from the start value given and read till the end value based on the increment used in the feature tracking module.

d. **OutputColormapFile**: Give the complete path where the colormap file created/modified in each timestep should be stored. The name of the colormap file is any valid text file name.

e. **InitialColorScheme**: User can either select 'Random' or 'UserSelectedFile'. If Random option is selected then 'TrackPloy' operator will randomly assign colors to all the objects in the first timestep selected. In subsequent timesteps each object will be assigned a color depending on the tracking information. If the user selects the 'UserSelectedFile' then the 'SelectedFile' attribute is enabled and user can give the complete path of the user defined colormap for the first timestep. 'TrackPloy' operator will assign colors to all the objects in the first timestep selected based on this colormap file. Again in subsequent timesteps each object will be assigned a color depending on the tracking information.

Click on the 'PlotAtts' and select the 'PolyData' plot to set the following attribute:

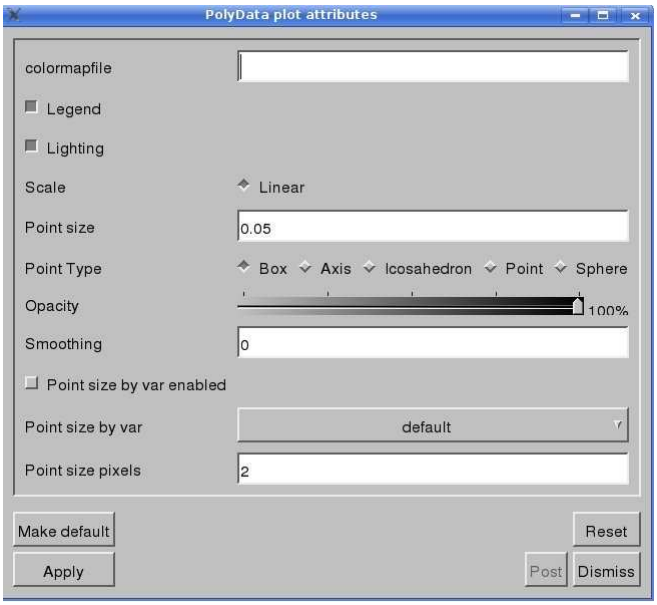


Figure 5.13: Plot Attributes of the 'PolyData' plot

a. **colormapfile**: Give the complete path where the colormap file created/modified by 'TrackPoly' is stored. The name of the colormap file should match the name of colormap file in the 'TrackPoly' operator attributes.

Execution

After selecting the plot and operator and setting their attributes, click on the 'Draw' button to begin visualization. When visualization is successfully performed, for each timestep 'TrackPoly' operator would write the colormap (object no, RGB value and transparency for each object). 'PolyData' operator would read the same map for each timestep and use it to display all objects in the given timeframe.

Source Code

The source code for this part is in three separate directories.

- Following is the path for the 'Poly' file reader database:
  $< visitpath > /src/databases/Poly.$
  When the file reader plug-in is created using XMLeditor, VisIt generates all the files required for correct loading of the plug-in. User has to modify the $avt < filereadername > FileFormat.Cfile$ to read the desired data type.

- Following is the path for the 'TrackPoly' operator:
  $< visitpath > /src/operators/TrackPoly/.$
  When the operator plug-in is created using XMLeditor, VisIt generates all the files required for correct loading of the plug-in. User has to modify the $avt < operatorname > Filter.Cfile$ to perform the desired filtering operation.

- Following is the path for the 'PolyData' plot:
  $< visitpath > /src/plots/PloyData/.$
  When the plot plug-in is created using XMLeditor, VisIt generates all the files required for correct loading of the plug-in. User has to modify the $avt < plotname > Filter.Candavt < plotname > Plot.Cfiles$ to perform the desired display operation.

See Appendix B for details of creating a new plug-in and using the XMLeditor.

Chapter 6

Results and Discussion

Feature tracking results are obtained for the following datasets:

1. Vorticity dataset: size 128*128*128 (75 timesteps)

2. Surfactant molecular dataset (Modeling and Simulation Department, P&G): size 47*47*47 (100 timesteps)

The feature tracking application is successfully ported to VisIt. The number of objects obtained in each timestep and their attributes are compared to the results obtained by AVS feature tracking utility. The object segmentation, quantification and tracking results in AVS and VisIt are identical.

Visualization is implemented as a separate module. Separation of Visualization from Feature Tracking enables the user to select any timestep to start the visualization process. Also, the user can select certain features to track by changing their color or by changing the transparency of the other undesired objects. This way user can highlight features of interest and fade other objects (completely or partially) in the background. Various tests were run to test the stability and correctness of the application. For each of the following tests performed, see section 5.1 to execute Feature Tracking and section 5.2 to execute visualization modules in VisIt.

6.1 Feature tracking and visualization of consecutive timesteps

Figure 6.1 shows the iso-surface visualization of 5 consecutive timesteps of vorticity data (available with the Vizlab). Each of the following dataset is of size 128*128*128:

1. vorts1.data

2. vorts2.data

3. vorts3.data

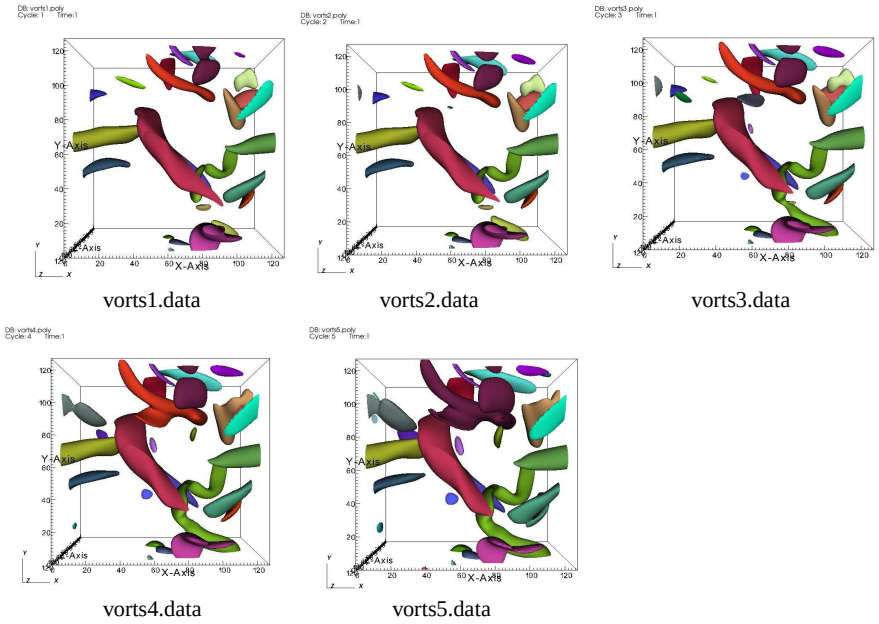


Figure 6.1: Iso-surface visualization of 5 consecutive timesteps. Each object of the first timestep vorts1.data is color-coded randomly. vorts2.data, vorts3.data, vorts4.data, vorts5. data have their objects color-coded based on tracking information.

4. vorts4.data
5. vorts5.data

First the feature tracking module is executed to generate the attribute files. The .poly file (contains iso-surface information) is used for visualization in the second part.

6.2 Feature tracking and selective visualization of consecutive timesteps

Figure 6.3 show the selective iso-surface visualization of 5 consecutive timesteps of vorticity data (available with the Vizlab). Each of the following dataset is of size 128*128*128:

1. vorts1.data
2. vorts2.data
3. vorts3.data
4. vorts4.data
5. vorts5.data

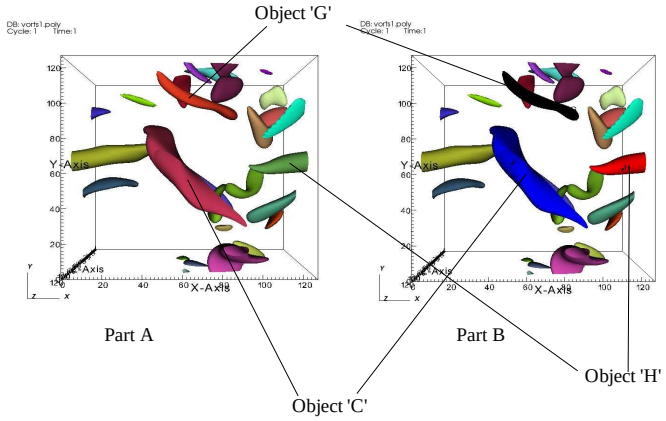


Figure 6.2: Selective Visualization: Part A - vorts1.data. is the first timestep, each feature is color-coded randomly. Part B - Color code of the three labeled objects is changed in the colormap file and visualization is performed again.

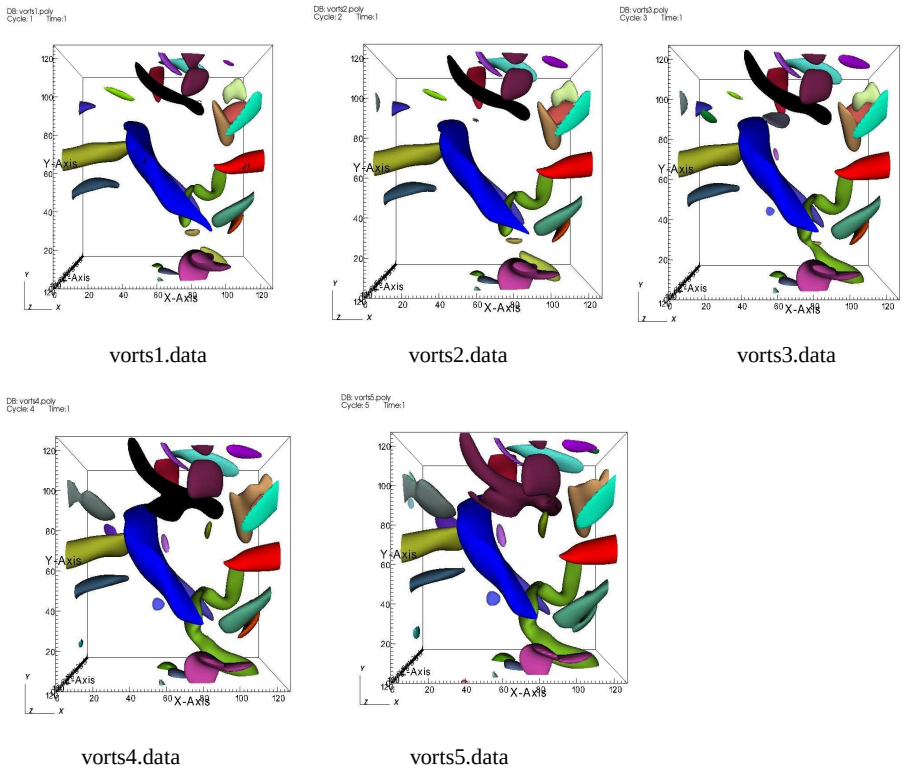


Figure 6.3: Selective iso-surface visualization of 5 consecutive timesteps. Colors of three objects in vorts1.data are modified as shown in Figure 6.2. vort2.data, vorts3.data, vorts4.data, vorts5.data have their objects colored as per tracking information

Figure 6.2 (part A), shows the objects in the first timestep generated by VisIt. Each of its features are randomly color-coded. In the same figure (part B) three objects (labeled as C,G and H) are of interest and should be tracked in subsequent timesteps, keeping the visual tracking of other objects intact. User can change the R,G,B color codes assigned to these objects in the colormap file and start the visualization process again (rest of the objects have the same color), this time making the colormap file selection as 'UserSelectedFile' (see section 5.1.1 to set operator attributes). Figure 6.3 shows the tracking of the 5 consecutive timesteps.

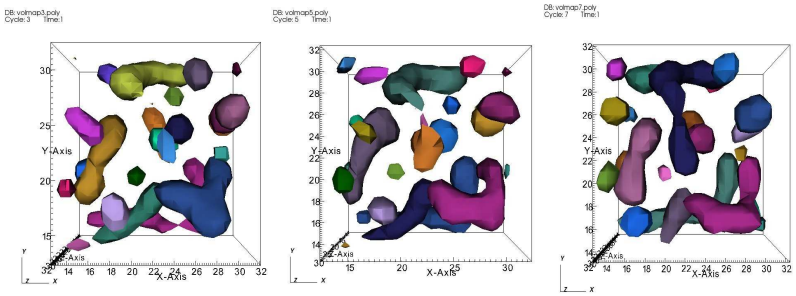
First the feature tracking module is executed to generate the attribute files. The .poly file (contains the iso-surface information) is used for visualization in second part.

Colors of all the objects in the first timestep can be changed. Such selective visualization helps to focus on evolution of specific features.

6.3 Feature tracking and visualization of non-consecutive timesteps

Figure 6.4 shows the iso-surface visualization of 3 non-consecutive timesteps of molecular data (available with Vizlab). Each of the following dataset is of size 47*47*47:

1. volmap3.bin
2. volmap5.bin
3. volmap7.bin



volmap3.bin volmap5.bin volmap7.bin

Figure 6.4: Iso-surface visualization of 3 non-consecutive timesteps. volmap3.bin is the first timestep hence each of its feature is color-coded randomly. volmap5.bin, volmap7.bin have their objects color coded as per tracking information

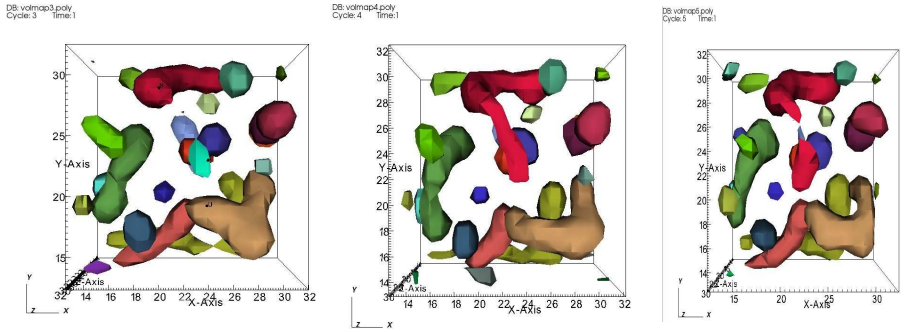
First the feature tracking module is executed to generate the attribute files. The .poly file (contains the iso-surface information) is used for visualization in second part.

Since VisIt tries to display any size of data on same size window, these images of smaller data size are not smooth.

6.4 Feature tracking and enhanced visualization: change transparency of the objects

Figure 6.5 show the iso-surface visualization of 3 consecutive timesteps of molecular data (available with Vizlab). Each of the following dataset is of size 47*47*47:

1. volmap3.bin
2. volmap4.bin
3. volmap5.bin



volmap3.bin volmap4.bin volmap5.bin

Figure 6.5: Iso-surface visualization of 3 consecutive timesteps. volmap3.bin is the first timestep hence each of its feature is color-coded randomly. volmap4.bin, volmap5.bin have their objects color-coded as per tracking information.

volmap3.bin is the first timestep and each of its features are randomly color-coded. Assume three objects (labeled as G,H and I) are of interest and should be tracked in subsequent timesteps. User can just fade the other object by changing the transparency of other objects in the colormap file and start the visualization process again, this time making the colormap file selection as 'UserSelectedFile' (see section 5.1.1: to set operator attributes). Figure 6.6 shows the tracking of the three selected objects and

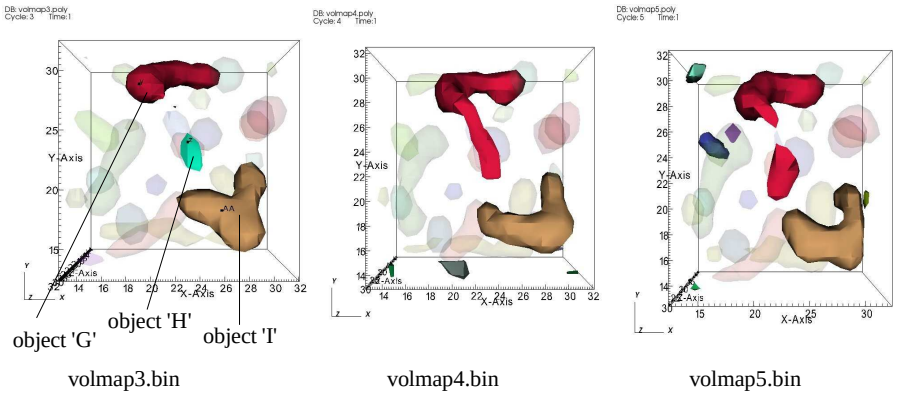


Figure 6.6: Enhanced iso-surface visualization: volmap3.bin is the first timestep hence each of its feature is color-coded randomly. Except the three labeled objects all objects have their transparency reduced. volmap4.bin, volmap5.bin have their objects color-coded as per tracking information.

other objects faded.

First the feature tracking module is executed to generate the attribute files. The .poly file (contains the iso-surface information) is used for visualization in second part.

Such kind of visualization reduces the visual clutter and allows to focus on specific features only.

As seen in the sections above, user can select the objects in each timestep and change their color codes for selective visualization. This is achieved by using the 'NodePick' option in VisIt. When a user activates this option and selects any object in the display window, VisIt picks up that node and shows the information as shown in the Figure 6.7. The node value corresponds to the object number and is used to change the color code in the colormap file. See Appendix A (A.2) for the format of the colormap file.

6.5 Execution time for Feature Tracking and Visualization modules

Table 6.1 gives the details of execution time of Feature Tracking test on vorticity and molecular datasets in VisIt. The test is run on consecutive timesteps without altering the color-code of any object manually. Test is run on machine1 with the following configuration:

Processor: Intel Core2 Duo

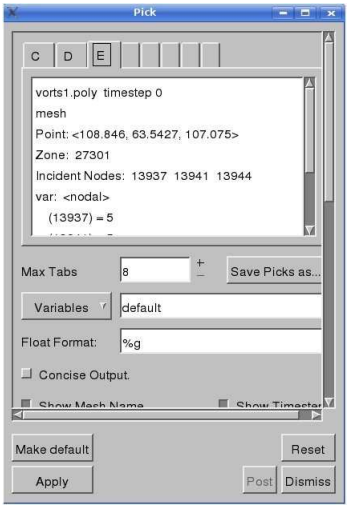


Figure 6.7: Three nodes are picked and labeled. Information of each node is shown. The node value is used to change the color- code in colormap file

RAM: 1GB

OS: Linux (Kubuntu)

Dataset	**NumTimesteps**	**Execute FeatureTrack**	**Execute Visualization**
vorts	75	1800 sec	250 sec
volmap	100	175 sec	115 sec

Table 6.1: Time to run Feature Tracking and Visualization for vorts and volmap dataset in VisIt on machine1

Table 6.2 gives the details of execution time of Feature Tracking test on vorticity and molecular datasets in VisIt. The test is run on consecutive timesteps without altering the color-code of any object manually. Test is run on machine2 with the following configuration:

Processor: Intel Pentium 4

RAM: 512MB

OS: Linux (Redhat)

Table 6.3 gives the details of execution time of the Feature Tracking test on vorticity and molecular datasets on AVS. The test is run on consecutive timesteps without altering

Dataset	NumTimesteps	Execute FeatureTrack	Execute Visualization
vorts	75	2520 sec	375 sec
volmap	100	234 sec	155 sec

Table 6.2: Time to run Feature Tracking for vorts and volmap dataset in VisIt on machine2

the color-code of any object manually. Test is run on machine2 with the following configuration:

Processor: Intel Pentium 4

RAM: 512MB

OS: Linux (Redhat)

Dataset	NumTimesteps	Execute FeatureTrack	Execute Visualization
vorts	75	2380 sec	355 sec
volmap	100	240 sec	155 sec

Table 6.3: Time to run Feature Tracking for vorts and volmap dataset in AVS on machine2

Table 6.2 and Table 6.3 show a comparison of execution time of Feature Tracking and Visualization on AVS and VisIt on the same machine configuration.

Chapter 7

Conclusion and Future Work

Feature tracking is an effective way to study the evolution of features over the time varying datasets. Vizlab at Rutgers has successfully developed the feature tracking utility in AVS, a commercial scientific visualization application. In this thesis the feature tracking utility is ported to VisIt, a free interactive scientific visualization tool. This would allow more scientists to use the feature tracking application. The feature tracking on VisIt was tested on different datasets and results match the AVS based Feature Tracking results. Separation of visualization from feature tracking allows the flexibility of selective visualization which helps to reduce the visual clutter.

Table 7.1 gives a comparison of various features of VisIt and AVS when developing or running feature tracking in VisIt and AVS.

Features	**VisIt**	**AVS**
Inherent data type used	vtkDataset (less flexible)	field Data (more flexible)
Input parameters	Input data files twice	Input data files only once
Non Consecutive timesteps	No additional handling required	Additional handling required
Single timestep	Not possible in Feature Tracking	Possible in Feature Tracking
Selective visualization	Allows to highlight any object	Allows to isolate only a single object
Changing colors of displayed objects	Possible	Not possible

Table 7.1: Comparison of the Features of VisIt and AVS

The current implementation for 'Feature Tracking' operator inherits from the 'avtTimeLoop' Filter to obtain information of the current timestep. This filter requires

atleast two timesteps to work correctly. This limitation can be possibly removed by writing a customized file reader for the '.bov' file which would provide the information to the 'FeatureTrack' operator about the current timestep, thus eliminating the need of 'avtTimeLoop' filter. This way Feature tracking module can be executed for a single timestep also. Use of such a file reader will also facilitate correctly writing the .visit file for grouping multiple timesteps. Currently, the .bov file reader guesses the sequence of timesteps selected for feature tracking, which at times requires re-ordering manually.

Current implementation handles only 3D rectilinear data type. This implementation can be extended to handle 2D structured data as well.

VisIt has the underlying capability to perform visualization on multi-processors or in distributed mode. The current implementation of feature tracking in VisIt does not utilize this powerful feature and hence it cannot be effectively used for very large datasets. Vizlab at Rutgers has developed a stand alone Feature Tracking utility which works in distributed mode [20] which can handle large datasets. The current feature tracking utility in VisIt can be made more powerful by extending it to work in the distributed mode which would allow the handling of very large datasets.

Appendix A

Format of Input/Output/Intermediate Files

A.1 Input/Output files created during the execution of Feature Tracking module

When the feature tracking module is executed, it reads the .bov files as input. .bov is a VisIt defined format for reading the information related to binary data files. Its format is as follows:

.bov file format:

TIME: $< give time to enable .bov reader to guess cycle. >$

DATA_FILE: $< name of the actual binary data file >$

DATA_SIZE: $< data extent in XYZ direction >$

DATA_FORMAT: $< data type which can be byte, short, floatetc >$

VARIABLE: $< variable name to identify the data during plotting >$

DATA_ENDIAN: $< BIG endian or LITTLE endian >$

BRICK_ORIGIN: $< Origin of the dataset in XYZ directions >$

BRICK_SIZE: $< Max extent of data in XYZ direction based on origin >$

Sample - .bov file:

TIME: 1.2345

DATA_FILE: vorts0.data

DATA_SIZE: 128 128 128

DATA_FORMAT: FLOAT

VARIABLE: vorts

DATA_ENDIAN: BIG

BRICK_ORIGIN: 0 0 0

BRICK_SIZE: 127 127 127

When the feature tracking module is executed, a 'GENERATED_TRACK_FILES' sub-folder is created under the directory from where the .bov files are read. The output attribute files generated are saved in this folder.

$< pathof.bovfiles >$GENERATED_TRACK_FILES/

Following output attribute files are written when the 'Feature Tracking' module runs:

1. **.poly file** - .poly file is written for each timestep. It contains node co-ordinates of the polygon vertices and their connectivity information for iso-surface visualization.
2. **.attr file** - .attr file is written for each timestep. It contains information about every object like centroid, moments, max node value, etc.
3. **.uocd file** - .uocd file is written for each timestep. It contains information pertaining to the frame. Some of the main points are No. of objects, Cell information.
4. **.trak file** - .trak file is written for each timestep. It contains number of objects and number of nodes to be used for memory allocation for tracking with successive timestep.
5. **.trakTable file** - A single .trakTable file is written which is updated in each timestep. This file contains the results of tracking. It gives the evolution history of each object frame by frame along with an indication of any events that occur.

A description of the file formats are as follows :

.poly file format:

$< red > < green > < blue >$

$< numnodes >$

$< x0 > < y0 > < z0 >$

$< x1 > < y1 > < z1 >$

$< x2 > < y2 > < z2 >$

$< x3 > < y3 > < z3 >$

$< x4 > < y4 > < z4 >$

.

.

.

$< numofconnections >$

3 $< vertexID >$ $< vertexID >$ $< vertexID >$

3 $< vertexID >$ $< vertexID >$ $< vertexID >$

3 $< vertexID >$ $< vertexID >$ $< vertexID >$

.

.

.

0

$< red >$ $< green >$ $< blue >$

Sample - .poly file :

0 158 96

6774

67.234009 102.000000 112.000000

68.000000 102.000000 111.367813

68.000000 101.106598 112.000000

.

.

.

13488

3 1 3 2

3 4 6 5

3 7 9 8

.

.

0

10 255 0

.

.

.attr file format:

object $< objID >$ attributes:

Max position: ($< maxX >$, $< maxY >$, $< maxZ >$) with value: $< maxnodevalue >$

Node No.: $< numnodemin >$

Min position: ($< minX >$, $< minY >$, $< minZ >$) with value: $< minnodevalue >$

Node No.: $< numnodemax >$

Integrated content: $< integratedcontent >$

Sum of squared content values: $< sumsquaredcontentvalue >$

Volume: $< volume >$

Centroid: ($< x >$, $< y >$, $< z >$)

Moment: Ixx = $< Ixx >$

Iyy = $< Iyy >$

Izz = $< Izz >$

Ixy = $< Ixy >$

Iyz = $< Iyz >$

Izx = $< Izx >$

Sample - .attr file:

object 0 attributes:

Max position: (63.000000, 56.000000, 127.000000) with value: 12.139389

Node No.: 2087999

Min position: (50.000000, 66.000000, 117.000000) with value: 6.073445

Node No.: 1925426

Integrated content: 52753.316406

Sum of squared content values: 420312.062500

Volume: 6857

Centroid: (54.960667, 66.905724, 118.313576)

Moment: Ixx = 81.173683

Iyy = 175.838699

Izz = 76.425446

Ixy = -105.836067

Iyz = -90.048668

Izx = 46.851215

---

object 1 attributes:

.

.

.uocd file format:

$< time >$

$< numobjects >$

$< objID >$ // starts from 0

$< volume >< mass >< centroidX >< centroidY >< centroidZ >$ // volume is the points number

$< Ixx >< Iyy >< Izz >< Ixy >< Iyz >< Izx >$

$< pointID >< pointX >< pointY >< pointZ >< pointvalue >$ // pointID starts from 0

$< pointID >< pointX >< pointY >< pointZ >< pointvalue >$

$< pointID >< pointX >< pointY >< pointZ >< pointvalue >$

.

. $< objID >$

.

.

Sample - .uocd file :

10.000000

38

0

12637 83432.195312 68.521065 97.046165 99.475235

234.301193 51.264282 245.409286 -85.529694 79.158249 -222.195129

1798593 65.000000 99.000000 109.000000 10.520865

.

. 1

.

.

The .trak file format :

$< filebasename >< time >< mass >< volume >< centroidX >< centroidY >$
$< centroidZ >$

$< filebasename >< time >< mass >< volume >< centroidX >< centroidY >$
$< centroidZ >$

.

.

.

Sample - .trak file :

/u61/rohinip/data/GENERATEDTRACKFILES/vorts10 10.000000 83432.195312 12637 68.521065 97.046165 99.475235

/u61/rohinip/data/GENERATEDTRACKFILES/vorts10 10.000000 42605.246094 6316 62.313812 117.506172 39.228920

/u61/rohinip/data/GENERATEDTRACKFILES/vorts10 10.000000 40073.558594 5771 62.039474 46.921017 4.576792

.

. **The .trakTable file format sample:**

In this file '-1' is used as delimiter to indicate an event

frame No.11

11 -1 13 26 // obj 11 of timestep 10 splits into 13 and 26 in timestep 11

19 21 -1 17 // obj 19, 21 of timestep 10 merge into obj 17 in timestep 11

1 -1 1 // obj 1 of timestep 10 continues as 1 in timestep 11

2 -1 2

.

.

.

-1 23 // new object 23 created in frame timestep 11

9 -1 // obj 9 of timestep 10 disappears in timestep 11

11 -1

Frame No.12

.

.

.

A.2 Input/Intermediate/Output files created during the execution of Visualization module

Visualization module reads the input .poly files (Format of .poly files is described in A.1). The file reader generates the intermediate 'curpoly.txt' file which the 'TrackPoly' operator reads to know the current .poly file being read for visualization. This 'curpoly.txt' file is created/updated for each timestep in the same folder where the input .poly files reside.

$< pathfor.polyfiles > /curpoly.txt$

curpoly.txt file format:

$< currentpolyfileread >$

Sample - curpoly.txt:

vorts1

The TrackPoly operator generates the intermediate 'colormap' file. This file contains the RGB color code for each object in the timestep as well as its transparency value. This file is read by the plot to create a color table. The file is created at the location given by the user in the 'TrackPoly' attributes.

$< colormapfile >$**.txt file format**:

$< objectnumber > < Rvalue(between0 - 255) > < Gvalue(between0 - 255) > < Bvalue(between0 - 255) > < transparencyvalue(between0 - 1) >$

Sample - curpoly.txt:

0 225 44 89 1

1 0 255 0 0.5

.

.

.

Appendix B

Installing VisIt

VisIt can be installed in two ways:

1. To use the existing VisIt functionalities without modifications:

For this purpose executables can be directly downloaded from VisIt homepage [9] and installed on the user machine.

2. To use existing VisIt functionalities as well as modify VisIt's source code:

For this purpose the user can download a script from VisIt homepage [9] and run it to install VisIt automatically on the user machine or download the VisIt source code and install VisIt manually by compiling the source code. Both of these installations allows the user to modify existing plugins, add new plugins etc.

Directory structure of VisIt source files:

Once the VisIt is installed on the user's machine, VisIt has a default directory structure as shown in the figure B.1.

Installing a new plugin in VisIt

User can add a database, operator or plot plugin to VisIt. Following are the steps to create a new operator plugin. Similar steps should be followed to create database or plot plugin:

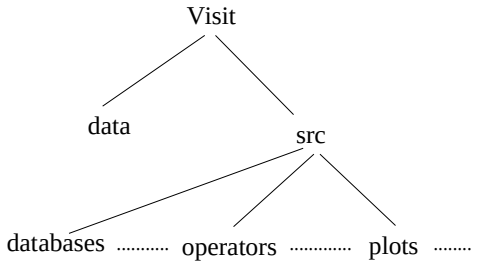


Figure B.1: The Default directory structure when VisIt is installed

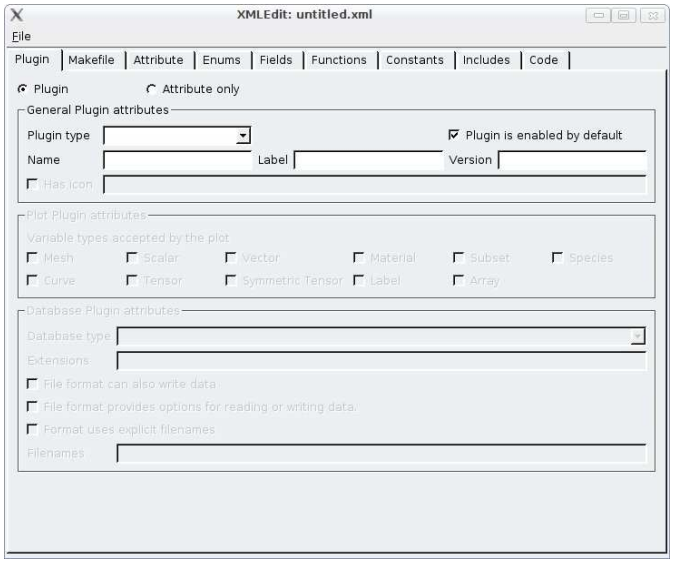


Figure B.2: xmlEditor allows the user to give information to create a customized plugin

1. Create a subfolder in the $< visitlocation/src >$/operator directory. The name of the subfolder should be same as the name of the new operator plugin.

2. VisIt makes the process of creation and compilation of plugins easier by providing the user with xml framework to create source code and makefile for the new plugin. On the command line, go to the folder $< newoperator >$. Type

$< visitpath/src/bin/ > xmledit < newoperator > .xml$

This will open an xml file for the user. Fill in the required fields and save the file. (Details about filling different fields is given in the VisIt developer's manual). Figure B.2 shows the xmleditor.

3. Then type $< visitpath/src/bin/ >$xml2plugin $< newoperator > .xml$

This command would internally invoke a series of commands to create source code and makefile for the new plugin as per the input in the ¡newoperator¿.xml file.

4. Type 'make'. When the user types make for the first time, VisIt gives an error in the 'avt$< newoperator >$Filter.C' file. This is expected since the user has to add code to make the plugin work. In other words, the steps mentioned till now will create a framework for the user to create a new plugin.

5. After making any changes to code, just type 'make'. Once the plugin compiles

successfully, go to the command line and type:

. < *visit*/*src* >/bin/visit

The new plugin created should be seen in the list of operators.

To add a plugin to VisIt created by another user using source code:

Assume that the plugin contains the 'FeatureTrack' operator.

1. Create a folder under < *visitpath* > /*src*/*operator* with the name FeatureTrack. Copy the source files to this folder.

2. cd FeatureTrack

3. < *visit* >/bin/xml2makefile FeatureTrack.xml

4. make

This should create and install the FeatureTrack plugin.

References

[1] X. Wang. *Visualizing and Tracking Features in 3D Time varying Datasets.* PhD thesis, Rutgers The State University, Elect & Comp. Engg. dept.

[2] Dhume Pinakin. Large scale feature extraction and tracking. Master's thesis, Rutgers The State University,Elect & Comp. Engg. dept., 2007.

[3] VisIt user reference, http://visitusers.org.

[4] An overview of AVT, the data flow network library used by VisIt's server, http://visitusers.org/index.php?title=Developer_Documentation.

[5] Image of VTK cells types, http://dunne.uni-hd.de/VisuSimple/documents/vtk-cell-types1.gif.

[6] Image of VTK grid type, www.cs.iupui.edu/ sfang/cs552/cs552-note2.pdf.

[7] D. Silver and Xin Wang. Tracking and visualizing turbulent 3d features. *IEEE Transactions on Visualization and Computer Graphics*, 3(2):129–141, April/June 1997.

[8] AVS homepage, http://www.avs.com.

[9] VisIt Homepage, https://wci.llnl.gov/codes/visit/.

[10] VTK homepage, http://www.vtk.org/.

[11] D. Silver and X. Wang. Tracking scalar features in unstructured data sets. In *Visualization '98. Proceedings*, pages 79–86, Research Triangle Park, NC, USA, USA, October 1998.

[12] Lian Jiang. PhD thesis, Rutgers, The State University, 2002.

[13] F. Reinders, F. H. Post, H. J. W. Spoelder, Visualization of Time-Dependent Data using Feature Tracking. ASCI'99 Conference Proceedings, M. Boasson, J.A. Kaandorp, J.F.M. Tonino, and M.G. Vosselman (eds.), 1999: p. 150-157.

[14] F. Reinders, F. H. Post, H. J. W. Spoelder, Attribute-Based Feature Tracking. Data Visualization '99: p. 63-72.

[15] G. Ji, H.W. Shen, R. Wenger, Volume Tracking Using Higher Dimensional Iscontouring. submitted to IEEE Visualization, 2003.

[16] R. Samtaney, D. Silver, N. Zabusky, J. Cao, Visualizing Features and Tracking Their Evolution. IEEE Computer, July 1994: p. 27(7):20-27.

[17] C. Bajaj, A. Shamir, B. S. Sohn, Progressive Tracking of Isosurfaces in Time-Varying Scalar Fields. CS and ICES Technical Report, University of Texas at Austin, 2002.

[18] D. Silver, Y. Kusursar, Visualizing Time Varying Distributed Datasets. Proceedings of Visualization Development Environment, 2000.

[19] C. Hansen, T. Udeshi, S. Parker, P. Shirley, Parallel methods for isosurface visualization, extended abstract. Ninth SIAM Conference on Parallel Processing for Scientific Computing, 1999.

[20] J. Chen, D. Silver, L. Jiang, The Feature Tree: Visualizing Feature Tracking in Distributed AMR Datasets. IEEE Symposium on Parallel and Large-Data Visualization and Graphics 2003: p. 103-110.

[21] M. Berger and J. Oliger, Adaptive Mesh Refinement for Hyperbolic Partial Differential Equations. Journal of Computational Physics, 1984: p. 484-512.

[22] D. Silver. Object-oriented visualization. *IEEE Computer Graphics and Applications*, 15(3):54–62, May 1995.

[23] Deborah Silver Xin Wang. Ocree based algorithm for three dimensional feature tracking. Technical report, Rutgers University,Elect & Comp Engg dept and CAIP, CoRE Building - Frelinghuesen Road, Rutgers University, Piscataway, NJ-08855, April 1996. CAIP-TR-204.

[24] Homepage of GGobi, http://www.ggobi.org/.

[25] Homepage of Paraview, http://www.paraview.org/New/index.html.

[26] VisIt feature description, https://wci.llnl.gov/codes/visit/about.html.

[27] Description: VisIt state objects, http://visitusers.org/index.php?title=State_objects.

[28] K. Martin B. Lorensen and W. Schroeder. *The Visualization Toolkit An Object-Oriented Approach To 3D Graphics*. Prentice Hall, 2nd edition edition, 1998.