FEATURE TRACKING & VISUALIZATION IN ‘VISIT’

By

NAVEEN ATMAKURI

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 2010
ABSTRACT OF THE THESIS

Feature Tracking & Visualization in VisIt

by Naveen Atmakuri

Thesis Director:
Professor Deborah Silver

The study and analysis of large experimental or simulation datasets in the field of science and engineering pose a great challenge to the scientists. These complex simulations generate data varying over a period of time. Scientists need to glean large quantities of time-varying data to understand the underlying physical phenomenon. This is where visualization tools can assist scientists in their quest for analysis and understanding of scientific data. Feature Tracking, developed at Visualization & Graphics Lab (Vizlab), Rutgers University, is one such visualization tool. Feature Tracking is an automated process to isolate and analyze certain regions or objects of interest, called ‘features’ and to highlight their underlying physical processes in time-varying 3D datasets.

In this thesis, we present a methodology and documentation on how to port ‘Feature Tracking’ into VisIt. VisIt is a freely available open-source visualization software package that has a rich feature set for visualizing and analyzing data. VisIt can successfully handle massive data quantities in the range of tera-scale. The technology covered by this thesis is an improvement over the previous work that focused on Feature Tracking in VisIt. In this thesis, the emphasis is on the visualization of features by assigning a constant color to the features (or objects) that move (or change their shape) over a period of time. Our algorithm gives scientists an option to choose only the features
of interest amongst all the extracted objects. Scientists can then focus their attention solely on those objects that could help them in understanding the underlying mechanism better. We tested our algorithm on various datasets and present the results in this thesis.
Acknowledgement

I would like to thank my advisor, Prof. Deborah Silver, for her support and encouragement while writing this thesis. Also, I would like to thank my parents and family who provided strong educational foundation and supported me in all my academic pursuits. I also acknowledge the help of VIZLAB at Rutgers.
# Table of Contents

Abstract ............................................................................................................................. ii

Acknowledgement ............................................................................................................ iv

List of Figures. .................................................................................................................... viii

1. An Overview of the thesis ......................................................................................... 1

2. Introduction to Scientific Visualization ................................................................. 5

   2.1 Examples of Scientific Visualization ................................................................. 7

   2.2 Scientific Visualization tools ............................................................................. 9

      2.2.1 VTK .............................................................................................................. 10

      2.1.2 AVS/Express ............................................................................................ 12

      2.1.3 ParaView .................................................................................................. 15

      2.1.4 VisIt .......................................................................................................... 17

3. Feature Tracking ....................................................................................................... 21

   3.1 Feature Tracking Algorithms at Vizlab ............................................................. 23

   3.2 Applications of Vizlab’s Feature Tracking Algorithms ..................................... 29

   3.3 Software Implementations of Feature Tracking Algorithms .......................... 31

      3.3.1 AVS/Express implementation ................................................................... 31

      3.3.2 Distributed Feature Tracking Implementation ......................................... 33

      3.3.3 VisIt Implementation ................................................................................ 34

4. VisIt Environment ..................................................................................................... 36
4.1 High level design of VisIt .................................................................37
4.2 Workflow of VisIt ............................................................................37
4.3 Plugin types in VisIt ........................................................................44
4.4 Adding new plugins in VisIt ............................................................45

5. Feature Tracking & Visualization in VisIt ...........................................50
  5.1 Motivation ........................................................................................50
  5.2 Feature Tracking & Visualization Functionalities ..........................52
    5.2.1 TrakTable based Color-coding ..................................................52
    5.2.2 Selective Feature Tracking .......................................................54
    5.2.3 Picking Objects by Mouse Click ................................................54
  5.3 Custom Plugins ...............................................................................55
    5.3.1 The Feature Extraction & Tracking group plugins ....................56
    5.3.2 The Visualization group plugins ..............................................56
    5.3.3 Auto-Generated Files ...............................................................57
    5.3.4 Newly added files ....................................................................59
  5.4 Feature Tracking & Visualization Workflow in VisIt .......................61
  5.5 Modifications to the Feature Tracking & Visualization Plugins ..........64
    5.5.1 The trakTable based coloring ....................................................65
    5.5.2 Selective Feature Tracking .......................................................71
    5.5.3 Picking Objects with a mouse click ..........................................74
  5.6 Technical Challenges with VisIt .......................................................75

6. Results .................................................................................................79

7. Conclusion ............................................................................................85
References .......................................................................................................................... 86

Appendix – I .......................................................................................................................... 89

Installation of the Feature Tracking & Visualization plugins in VisIt .................................. 89
Feature Tracking & Visualization User manual ..................................................................... 90

Appendix – II .......................................................................................................................... 100

File Formats ......................................................................................................................... 100
Data structures in Scientific Visualization ......................................................................... 110
Pick Modes .......................................................................................................................... 113
AVS to vtk converter ........................................................................................................... 114
2D to 3D converter ............................................................................................................. 117

Appendix – III ........................................................................................................................ 118

Feature Tracking related publications ................................................................................ 118
List of figures

Figure 1: Visualization of an array of number as an image……………………………5
Figure 2: Examples of visualizations in the study of natural sciences. .....................7
Figure 3: Visualization application in geography and ecology. .............................8
Figure 4: Examples of visualizations in formal sciences. .................................8
Figure 5: Examples of visualizations in Applied Sciences. .................................9
Figure 6: An example of AVS Network. ..............................................................14
Figure 7: VisIt Dataflow network. .................................................................20
Figure 8: The Feature Tracking on hurricane data. .............................................21
Figure 9: The Feature Tracking based visualization pipeline. ............................24
Figure 10: The Featuring Tracking Interactions. ..............................................25
Figure 11: The overlap based Feature Tracking algorithm using Octree datastructure. 26
Figure 12: The Feature Tracking on cloud water simulation. ..............................30
Figure 13: The Feature Tracking of a simulation of the decay of isotrophic
tubulence in a box (unstructured dataset), 500 timesteps total. .........................31
Figure 14: VisIt High level design. .................................................................38
Figure 15: The flow diagram for the process associated with loading of a data
file in VisIt. .....................................................................................................40
Figure 16: xmlEdit snapshot ...........................................................................47
Figure 17: The trakTable based Coloring. .........................................................53
Figure 18: The Feature Tracking Plugin development in VisIt. .........................55
Figure 19: The Feature Tracking & Visualization process in VisIt. ...................62
Figure 20: A Tree list view of methods added to the Feature Tracking &
Visualization in VisIt..................................................................................64

Figure 21: Selective Feature Tracking on the Vorts dataset. .........................71

Figure 22: Automatic database creation by VisIt. ........................................78

Figure 23: The visualization results for the Feature Tracking on the vorticity

dataset (vorts).............................................................................................80

Figure 24: Selective Feature Tracking of Object Number 45 in the DNS dataset. ....81

Figure 25: Results from the application of the Feature Tracking & Visualization

plugins on the volmap dataset.......................................................................82

Figure 26: Feature Tracking & Visualization results from four different frames

of the Indicator (2D) dataset. ........................................................................83

Figure 27: Selective Feature Tracking of 2 objects from the Indicator (2D) dataset. ....84

Figure 28: The FeatureTrack operator attributes screenshot. .................................91

Figure 29: The PolyData and TrackPoly Attributes screenshot. .............................93

Figure 30: Visualizing the .poly file created by the ‘Feature Extraction & Tracking’

module.......................................................................................................94

Figure 31: The ‘TrackPoly’ operator and the ‘PolyData’ plot attribute window. ........96

Figure 32: Selective Feature Tracking and Visualization of a single object

(Object number 5) from the vorticity dataset (vorts), frames 1 to 6. ..........97

Figure 33: Picking objects with mouse clicks. .................................................98

Figure 34: Pick Information window ................................................................99

Figure 35: An Uniform Mesh.........................................................................110

Figure 36: A Rectilinear Mesh. .....................................................................111

Figure 37: An Irregular Mesh. .......................................................................111
Figure 38: A Structured Mesh. ................................................................. 112

Figure 39: An Unstructured Mesh. ......................................................... 112
Chapter 1

An Overview of the thesis

The process of converting raw data into a form that is viewable and understandable to human beings is called visualization. Visualization allows us to get a better cognitive understanding of the data. Visualizations dealing with the scientific data that have a well-defined representation (in 2D or 3D) or have a natural geometric structure (e.g., MRI data or wind flows) are categorized as Scientific Visualizations. Scientific visualizations provide a more intuitive interpretation for the process of hypothesis building and modeling. Examples of scientific visualizations are the visualizations of intense quantities of laboratory or simulation data, or the results from sensors out in the field. The output of these simulations, experimental and sensor data are in the form of 3D scalar datasets. In addition, time-varying simulations are common in many scientific domains; these are used in the study of evolution of phenomena or features. Traditionally, visualization of 3D time-varying datasets is done using animation, i.e., each frame (or time-step) is visualized using iso-surfacing or volume rendering, and then the various time-steps are run in sequence to decipher any visual patterns. However, for datasets with continuously evolving features like cloud formations, it is difficult to follow and see patterns in 3D. What is required is a technique to isolate and analyze certain regions of interest also called 'features' and highlight their underlying physical processes [1, 2]. For example, it is usually important to know what regions are evolving, whether they merge with other regions, and how their volume may change over time. Therefore, region based approaches, such as Feature Tracking and Feature Quantification are needed. Moreover, most of the standard visualization methods cannot give a quantitative description of the
evolving phenomena, such as the centroid of a merged region or the value of its moments. An automated procedure to track features and to detect particular stages or events in their evolution can help scientists concentrate on regions and phenomena of interest. Color connected iso-surface based on computed quantification provide visual cues about the events or particular stages of interest. This effectively reduces the amount of data to focus on by curtailing the visual clutter. Another important application of Feature Tracking is in data mining. By building a database of features, the results over multiple simulations can be used for ’event matching’. In previous work [3,4,5], the Vizlab had pioneered the use of Feature Tracking to effectively visualize time-varying datasets. Feature Tracking was implemented as a plugin on AVS, a proprietary visualization software. Thus license costs limit the usability and extendibility over a larger user base.

Many open-source visualization software packages that contain lot a of features and functionalities that are available to users free of cost. VisIt is one such open-source visualization software that is freely available and has a rich feature set. VisIt supports quantitative and qualitative visualization, it has powerful user interface and architecture to support massive datasets in the order of tera scale. VisIt allows development of custom plugins to add new features and functionality. Feature Tracking capability can be introduced in VisIt as plugins, and this would allow VisIt user to study evolving patterns in time-varying datasets.

Previous work on porting Feature Tracking to VisIt [23] extracted the features from a time-varying scalar dataset. However in [23], the visualization plugins were found to be
incomplete and incompatible with newer versions of VisIt. If an object is moving across a dataset changing its shape and size, the algorithm should be able to identify this phenomenon and assign the same color to object in other frames until the object disappears. If an object splits into smaller objects, all the smaller objects should have the same color as parent object. This kind of behavior is captured in our algorithm and we use that information to assign colors and achieve our task. Our implementation gives the scientist or a user an option to selectively track only few features (or objects) of interest amongst all the extracted features. These features can be selected by mouse clicks. Like most open-source softwares, VisIt too lacks proper design documentation. This hampers the development of new features or plugins in VisIt. A lot of time and energy is spent in understanding the design, so this thesis also aims to document the design decisions that could be useful to other developers.

The organization of the thesis is as follows: Chapter 2 provides an introduction to Scientific Visualization and describes some software packages like AVS, ParaView and VisIt. Chapter 3 describes Feature Tracking process and various software implementations that exist in Vizlab. Chapter 4 is about VisIt. We discuss the design, working and procedure for creating new plugins in VisIt. In Chapter 5, we talk about the functionalities that have to be added to VisIt, design of Feature Tracking & Visualization plugins, new methods added, modifications made to the existing code and the challenges faced during this process of designing new plugins. The algorithm was tested on different datasets and results presented in Chapter 6. Finally thesis concludes with reiterating the usefulness of Feature Tracking in VisIt and its benefits as a Scientific Visualization tool.
We provided a detailed user manual about the installation procedure and information on using the plugins in appendix.
Chapter 2

Introduction to Scientific Visualization

Visualization is the process of presenting data in a form that allows rapid understanding of relationships and findings that are not readily evident from raw data. In Figure 1, we have a raw data file that represents an array of RGB color values. It’s very hard to these interpret the meaning of these numbers on their own, but when presented as a picture we get a better idea of what the data is representing.

As stated in [8], Scientific Visualization is an interdisciplinary branch of science, “primarily concerned with the visualization of three dimensional phenomena (architectural, meteorological, medical, fluid flow, biological etc), where emphasis is on realistic rendering of volumes, surfaces, illumination sources, and so forth, perhaps with a dynamic component of time”. Scientific Visualization is the use of data driven computer graphics to aid in the understanding of scientific data. Is Scientific Visualization just computer graphics, then? Computer graphics is the medium in which modern
visualization is practiced, however visualization (including Scientific Visualization) is more than simply computer graphics. It uses graphics as a tool to represent data and concepts from computer simulations and collected data. Visualization is actually the process of selecting and combining representations and packaging this information in understandable presentations. Scientific Visualization is a form of communication and we need to be very clear of our audience, as they have to grasp what happens to the information as it passes from numbers to pictures. For instance, the goal might be to demonstrate a scientific concept, in which case the presentation to scientist would be different from a presentation that would be shown to the general public. The amount and level of detail required in visualization is based on experience of the intended audience.

At one level, Scientific Visualization can be thought of analytically as a transfer function between the numbers and images. At another level, visualization involves a barrage of procedures, each of which may influence the final outcome and the ability to convey meaningful information. The process of visualization roughly consists of the following steps:

- **Data Filtering** – this step cleans and processes the data to yield meaningful results. Examples of Data Filtering are: removing noise, replacing missing values, clamping values in a certain range or producing new numeric forms leading to greater insights about the data.

- **Representation** – Visual representation of information requires certain literacy on the part of the developer and the viewer [9]. Beyond numerical representation of the output of the simulation, it’s advisable to give information about the simulation itself, for e.g., the grid of the computation domain, coordinate system, scale information and resolution of computation. The goal of visualization limits
the medium of delivery, which in turn puts constraints on the possible choices of representations. So, for example, if the motion is an important aspect to show from the data, then a medium that supports time-varying imagery should be used.

- **Feedback** – It is a good practice for scientists to question the accuracy and the validity of all the information that is presented to them. It’s always important to get this feedback and make changes to the process in order to get proper and accurate results in visualization for the intended audience.

### 2.1 Examples of Scientific Visualization

Visualization tools when applied to scientific data produce beautiful insights into many natural phenomena. In this section, we see the results from application of visualization tools in various scientific domains:

**Natural science** – In Natural sciences visualizations could be useful in studying star formations, understanding gravity waves, visualizing massive supernova explosions and molecular rendering. Figure 2 shows some of these results.

![Visualization Examples](https://wci.llnl.gov/codes/visit/gallery.html)

Figure 2: Examples of visualizations in the study of natural sciences. a) Star formation. b) Gravity plot. c) Visualization of massive supernova explosion. d) Molecular rendering. (Images Source - https://wci.llnl.gov/codes/visit/gallery.html)

**Geography and Ecology** – In geography and ecology, visualization tools are useful for climate visualization, terrain rendering and studying the atmospheric anomalies in areas
like Times Square. Figure 3 shows the visualizations of the concepts from field of Geography and Ecology.

![Figure 3: Visualization application in geography and ecology. a) Visualization of terrain. b) Climate Visualization. c) results from simulation framework of atmospheric anomaly around Times Square. (Images source - https://wci.llnl.gov/codes/visit/gallery.html)](image)

**Formal sciences** – In formal sciences, visualization tools can benefit users by showing a mapping of topographical surfaces, representing huge quantities of data in curve plots or scatter plots. In Figure 4, we can see that Image annotation is also a technique in this category.

![Figure 4: Examples of visualizations in formal sciences. a) Curve plot b) Image annotations. c) Scatter plot. (Images source - https://wci.llnl.gov/codes/visit/gallery.html)](image)

**Applied sciences** – Visualization tools are very useful for manufacturing and automobile industry. These tools reveal a lot of information about the design of cars and aircrafts without actually manufacturing them thus saving a lot of money. These tools are used to model cars, study the aerodynamics of an aircraft, and render traffic measurement in the city for city planners to come up with effective traffic management solutions.
2.2 Scientific Visualization tools

A number of tools are available for creating visualizations of scientific data. These tools can be categorized as:

- **Visualization libraries** - Software libraries were developed that enabled researchers to generate charts, graphs and plots without the need for reinventing the graphics themselves. Since, the form of interaction is through programming, it has limited interactivity. VTK[13] is a widely used C++ class library for 3D graphics and visualization that is freely available.

- **Turn-key packages** – are visualization packages (or programs) designed specifically for visualization purposes. Turn-key packages contains controls (widgets) for most options that users would want to exercise when visualizing data. This is accomplished through the use of pull down menus or popup windows with control panels. Examples are Vis-5D [39], Gnuplot [40] etc.

- **Dataflow packages** – These are designed as tools to be used directly by the scientist. The dataflow concept consists of breaking down the tasks into small
programs, each of which does one thing. Each task is represented as a module. Examples of this kind of packages are softwares like AVS [21].

- **Writing Your Own Software (WYOS)** - before Dataflow packages and other tools were available, the programs were customized for a particular task in hand. This is sometimes still done with large time-varying datasets, but now mostly people use off-the-shelf software packages with some modifications to perform a particular task.

Visualization software packages (sometimes referred to as dataflow packages) are the most prominently used visualization tools. These software packages provide users a plethora of options to quickly generate visualizations from the raw data, animate them through time, manipulate them and save the resulting images for presentations. These visualization software packages are modular, based on Object Oriented principles that facilitate addition of new capabilities as modules. Some of the visualization software packages provide tools to the developers that make process of adding new modules much simpler. These tools generate an automated framework so that new modules can be easily integrated into the visualization software package. Developers just need to implement the algorithm for new modules in some high level programming language like C++, Java, or Python. In this thesis, we discuss 4 visualization tools: AVS, VisIt, ParaView. We also give a brief documentation of VTK. Both ParaView and VisIt are written using VTK.

### 2.2.1 VTK

The Visualization ToolKit (VTK) is an open source, freely available software system for 3D computer graphics, image processing, and visualization that is used by thousands of researchers and developers around the world. VTK consists of a C++ class library, and
several interpreted interface layers including Tcl/Tk, Java, and Python. VTK supports a wide variety of visualization algorithms including scalar, vector, tensor, texture, and volumetric methods; and advanced modeling techniques such as implicit modeling, polygon reduction, mesh smoothing, cutting, contouring, and Delaunay triangulation. In addition, dozens of imaging algorithms have been directly integrated to allow the user to mix 2D imaging / 3D graphics algorithms and data. VTK has the following attractive features:

- **Toolkit philosophy** – toolkits enable complex applications to be built from small pieces and the smaller pieces have simpler interfaces. In this way, smaller pieces can be readily assembled into larger systems.

- **Interpreted Language interface** - interpreted applications can be built significantly faster than compiled languages mainly because of elimination of compile /link cycle. Also interpreted languages are written at a higher level languages resulting in simpler more compact code that is faster to write and debug. The core computational objects are built using compiled languages whereas higher level applications are built with interpreted languages.

- **Extremely simple** – encouraging wider use of visualization and graphics. This philosophy benefits by reducing the effort to maintain, extend and interface the toolkit.

Object Oriented design of VTK has two distinct parts. The first is the graphics model that is an abstract model for 3D graphics. The second is the visualization model that is dataflow model of the visualization process. The graphics model captures the essential features of a 3D graphics system in a form that is easy to understand and use. There are nine basic objects in the model. These are: Render Master, Render Window, Renderer,
Light, Camera, Actor, Property, Mapper and Transform. The visualization model is based on the dataflow paradigm adopted by many commercial systems. In this paradigm modules are connected together into a network. The modules perform algorithmic operations on the data as it flows through the network. This model has two types of objects: process objects and data objects. Process objects are the modules or algorithm portions of the visualization network. Data objects, also referred to as datasets, represent and enable operations on the data that flow through the network. Process objects may further be classified into one of three types: sources, filters and mappers. Source objects initiate the network and generate one or more output datasets. Filters require one or more inputs and generate one or more outputs. Mappers that require one or more inputs, terminate the network.

2.1.2 AVS/Express

AVS/Express is a comprehensive and versatile data visualization tool for both non-programmers and experienced developers. AVS provides powerful visualization methods for challenging problems in science, business, engineering, medicine, telecommunications and environmental research. AVS/Express enables object-oriented development of rich and highly interactive scientific and technical data visualizations for a wide range of computing platforms. [21]. AVS/Express has the following attractive features:

• **Object Oriented** - AVS/Express' development approach is object-oriented; it supports the encapsulation of data and methods, class inheritance, templates and
instances, object hierarchies and polymorphism. In AVS/Express, all application components, from the lowest to the highest level, are objects.

- **Visual development** - The Network Editor is AVS/Express' main interface. It is a visual development environment that is used to connect, define, assemble, and manipulate objects through mouse-driven operations.

- **Visualization application** - AVS/Express provides hundreds of predefined application components (objects) that process, display, and manipulate data. The objects and application components that you connect and assemble in the Network Editor control how data is processed and how it is displayed.

Furthermore, AVS/Express also provides programming interface (APIs) to C, C++ and Fortran, allowing developers to easily integrate their own modules into AVS/Express.

**From data to pictures in AVS.**

To transform the data into pictures in AVS, one must follow these steps:

1. Import the data in AVS
2. Process the Data, if needed
3. Apply one or more Visualization techniques.
4. View the results.
AVS has many built-in modules for performing the above-mentioned tasks. For example, the ‘ReadField’ module imports data from .fld file (see Appendix – I for the format of the file) into AVS; the ‘downsize’ module processes the data and as the name suggests, it downsizes the data based on some criteria provided by the user. There are modules to apply visualization techniques and view the results. A user selects the appropriate modules manually and builds the network by connecting different modules as shown in Figure 6. AVS facilitates the process of network building by color-coding the input and output ports. The input port of one module can be connected to the output port of another module having the same color. Figure 6, shows a simple network in AVS to read field files, downsize the data according to some criteria and produce the orthoslices for the volume. In Figure 2, the modules ‘bounds’ and ‘Axis3D’ show a bounding box and the axis for the volume. The ‘UViewer3D’ module renders the 3D volume corresponding to the input data on the screen. A change in the input parameters for any of the modules causes the entire network to be executed again.
2.1.3 ParaView

ParaView is an open-source, multi-platform data analysis and visualization software package. With ParaView, users can quickly build visualizations to analyze their data using qualitative and quantitative techniques. The data exploration can be done interactively in 3D or programmatically using ParaView's batch processing capabilities. ParaView has been successfully tested on Windows, Mac OS X, Linux, IBM Blue Gene, Cray Xt3 and various Unix workstations, clusters and supercomputers. Under the hood, ParaView uses the Visualization Toolkit (VTK) [13] as the data processing and rendering engine and has a user interface written using Qt. Some of the important features of ParaView are given below:

- **Visualization Capabilities** – ParaView handles structured, unstructured, polygonal, multiblock and AMR data types. All processing (or filtering) operations produce datasets. ParaView can be used to inspect vector field by applying glyphs, extract contours and iso-surfaces, cut or clip regions by clipping planes, or generate streamlines using constant step or adaptive iterators. The points in a dataset can be warped with scalar or vector quantities. Python programming interface can be used for advanced data processing.

- **Input/Output and file formats** - ParaView supports a variety of file formats. Since, it's based on VTK, it supports most of the file formats supported by VTK.

- **User interaction** – Qt application framework introduces flexibility and interactivity. Parameters on the filters can be changed by directly interacting with the 3D view using 3D manipulators. Interactive frame rates in maintained by using LOD (level of detail) models.
• **Large data and distributed computing** – ParaView runs parallelly on distributed and shared memory systems using MPI. These include workstation clusters, visualization systems, large servers, supercomputers, etc. ParaView uses a data parallel model in which the data is broken into pieces to be processed by different processes. Most of the visualization algorithms function without any change when running in parallel. ParaView supports distributed rendering (where the results are rendered on each node and composited later using the depth buffer), local rendering (where the resulting polygons are collected on one node and rendered locally) and a combination of both (for example, the level-of-detail models can be rendered locally whereas the full models are rendered in a distributed manner). This provides scalable rendering for large data without sacrificing performance when working with smaller data.

• **Scripting and extensibility** – ParaView is fully scriptable using simple but powerful Python language. Additional modules can be added by either writing an XML description of the interface or by writing C++ classes. The XML interface allows users/developers to add their own VTK filters to ParaView without writing any special code and/or re-compiling.

**From data to pictures in ParaView**

The procedure for converting data to pictures in ParaView is similar to that of AVS. First, the data has to be read in ParaView. Since, ParaView is based on VTK, it supports most of the file formats supported by VTK. If ParaView is unable to find a reader associated with a particular file format, then an additional reader module has to be written to read the data in ParaView. Once the data is read in ParaView, the surface is rendered on the
screen as a solid mesh. However, interesting features cannot be determined by simply looking at this solid mesh. There are many variables associated with the mesh (scalars and vectors) and mesh being a solid hides a lot of information inside it. We can discover more information about the data by applying Filters. Filters are functional units that process the data to generate, extract, or derive features from the data. Filters are attached to readers, sources, or other filters to modify its data in some way. These filter connections form a visualization pipeline. Filters forming a visualization pipeline are selected by choosing corresponding icons on the filter toolbar. ParaView automatically creates this pipeline; unlike AVS, users need not worry about connecting the individual modules. Once, user has finished selecting the appropriate filters, the results are rendered on the screen by clicking “Apply” Button.

### 2.1.4 VisIt

VisIt [11] is an open source, turnkey application for large scale simulated and experimental data sets. VisIt is an infrastructure for parallelized and general post-processing of extremely massive data sets. Target use cases for VisIt include data exploration, comparative analysis, visual debugging, quantitative analysis, and presentation graphics. VisIt leverages several third party libraries like: the Qt widget library [12], the Python programming language and the Visualization ToolKit (VTK) library [13] for its data model and many of its visualization algorithms. VisIt has been ported to Windows, Mac, and many UNIX variants, including AIX, IRIX, Solaris, Tru64, and, of course, Linux, including ports for SGI's Altix, Cray's XT4, and many commodity clusters. Some of the key features of VisIt are listed below:
• **Rich set of features for scalar, vector and tensor visualization** – VisIt’s visualization options can be broadly classified in two main categories (as mentioned in the VisIt Developer Manual [20]):

  1. **Plots** – to visualize data and include boundary, contour, curve, mesh, streamline, subset, surface, tensor, vector.

  2. **Operators** – consists of operations that can be performed on the data prior to visualization, like slice, index, onion peel, iso-surface etc.

• **Qualitative and Quantitative visualization** - VisIt is also a powerful analysis tool. It provides support for derived fields that allow new fields to be calculated using existing fields. For example, if a dataset contains a velocity field, it is possible to define a new field that is the velocity magnitude.

• **Supports multiple mesh type** - VisIt provides support for a wide range of computational meshes, including two- and three-dimensional point, rectilinear, curvilinear, and unstructured meshes. In addition, VisIt supports structured AMR meshes and CSG meshes.

• **Powerful full featured Graphical User Interface (GUI)** - VisIt’s graphical user interface allows novice users to quickly get started visualizing their data, as well as allowing power users access to advanced features. VisIt automatically creates time-based animations from data sets that contain multiple time steps.

• **Parallel and distributed architecture for visualizing tera-scale data** - VisIt employs a distributed and parallel architecture in order to handle extremely large data sets interactively. VisIt’s rendering and data processing capabilities are split into viewer and engine components that may be distributed across multiple machines.
• **Interfaces with C++, Java and Python** - VisIt also supports C++, Python and Java interfaces. The C++ and Java interfaces make it possible to provide alternate user interfaces for VisIt or allow existing C++ or Java applications to add visualization support. The Python scripting interface gives users the ability to batch process data using a powerful scripting language.

• **Extensible with dynamically loaded plug-ins** - VisIt achieves extensibility through the use of dynamically loaded plugins. All of VisIt’s plots, operators, and database readers are implemented as plugins and are loaded at run-time from the plugin directory.

**From data to pictures in VisIt.**

In VisIt too, the process of transforming the data to pictures is similar to other visualization software packages. The difference comes in the underlying network that is created. VisIt automatically creates an AVT network for the user, depending on the action performed by the user. If the user performs the following actions:

- Load a data file
- Apply Operator (or filter) to the data, if any. And user chooses splice operator
- Choose a plot (contour plot).
- Execute the network and draw the results on the screen.
Then a network as shown in Figure 7 is generated automatically by VisIt. This network shows contribution of each user action in building the network. When the user chooses “Apply”, the network is executed and results are rendered on to the ‘Viewer’ (screen).

VisIt provides plugins corresponding to each of the action. There are many built-in plugins for handling database (i.e loading files), operators and plots. Custom plugins have to be developed for reading file formats not supported by VisIt or adding new capabilities to plots and operators. In this thesis, we continue the development based on VisIt and more information about the working of VisIt, its internal details and the dataflow network is given in Chapter 4.
Chapter 3

Feature Tracking

Most complex simulations and observations generate data over a period of time. Such time-varying data have one or more coherent amorphous regions, called features [27] that might be of interest to the scientists or the users. Feature Tracking tracks the features that play an important role in studying the evolution of different physical phenomena. Feature tracking can be very useful in analysis of hurricanes and development of a prediction system to minimize the damage caused by the hurricanes. Figure 8 shows the results from application of the Feature Tracking technique on hurricane data from ‘Hurricane Bonnie’ [29]. In Figure 8: (a) one feature was tracked for 30 frames (or timesteps), in figure 8(b) three independent features were tracked for 30 frames and in figure 8(c) a number of independent features were tracked over the same time period of 30 frames. From these results it was possible to see that most of the features under consideration followed a clear pattern such as moving counter clockwise and inwards. Also, one could see features closer to hurricane’s center moving faster than the features farther from the hurricane’s eye. Both of these findings were extremely important in terms of the analysis and interpretation of hurricane data.

![Figure 8: The Feature Tracking on hurricane data. a) Results of tracking a hurricane feature within 30 timesteps. b) Path followed by three independent features over the same period of time. C) Resulting path after tracking a number of independent features over time (Image source – [29])](image-url)
Over the years, researchers have proposed several different techniques for extracting and tracking features from time-varying datasets. These techniques can be broadly classified as:

1. **Overlap based Feature Tracking** - Silver & Wang [30, 4] presented a feature tracking technique that extracts features, organizes them into an octree structure and tracks the threshold connected components in subsequent time frames by assuming that all the features between adjacent time frames overlap. Later on, the authors implemented an algorithm for tracking features in unstructured datasets.

2. **Attribute based Feature Tracking** - Reinders et. al [31] described a tracking technique that tracks features based on attributes like mass, position and size. A number of correspondence functions are tested for each attribute resulting in correspondence factor. Correspondence factor makes it possible to match likely feature across subsequent frame in a dataset.

3. **Higher dimensional isosurfacing based Feature Tracking** – Ji et al [32] introduced a method to track local features from time-varying data by analyzing higher dimensional geometry. Rather than extracting features such as isosurfaces separately from individual frames of a dataset and computing the spatial correspondence between the features, correspondence between the features can be obtained by higher dimensional isosurfacings.

4. **Machine learning based Feature Tracking** - Tzen and Ma [33] presented a machine learning based visualization system to extract and track features of interest in higher dimensional space without specifying the relations between those dimensions. The visualization system “learns” to extract and track features in complex 4D flow field according to their “visual properties” like the location, shape
and size.

5. **Time Activity Curve (TAC) based Distance Fields for time-varying Features** – Lee and Shen [41] presented a new framework for visualizing time-varying features and their motion without explicit feature segmentation and tracking. A time series or Time Activity Curve (TAC) represents time-varying features. Dynamic Time Warping (DTW) distance metric is used to compute similarity, between a voxel’s time series and the feature. The purpose of DTW is to compare the shape similarity between two time series with an optimal warping time so that phase shift of the features in time can be accounted for. Based on the TAC-based distance field, several visualization methods can be derived to highlight the position and motion of the features.

3.1 **Feature Tracking Algorithms at Vizlab**

At Vizlab, the Feature Tracking algorithms for 3D Scalar Fields are based on a framework as shown in Figure (9). The goal of the process is to obtain dramatic data reduction and thus help scientists quickly focus on a few features or events of interest. The major steps are:

- **Feature Extraction** - The first step is to identify and segment features of interest from the dataset to be tracked. The method used depends on the definition of a ’feature’, which can differ from domain to domain. Usually features are defined as threshold-connected components [6,7,3].
Figure 9: The Feature Tracking based visualization pipeline.
Image source - [6]

- **Feature Tracking** - In this step, the evolution of the extracted features is followed over time noting various events that occur during their evolution.

- **Feature Quantification** - Once features are extracted, they are quantified and information about them, e.g., mass, centroid, etc. can be calculated.

- **Enhanced visualization and event querying** - Using the accumulated tracking information, we can also perform additional visualization steps like event querying which involves gathering information leading to a certain event of interest or present a new visualization using the data (metadata) collected. One example of this is volume rendering of an individual feature.

Over the years, various students have made refinements to the Feature Tracking algorithms as a part of their thesis work. Please refer to Appendix – III for the list of Feature Tracking related publications. First, we describe the basic algorithm and then the refinements.
3.1.1 Overlap based Feature Tracking Algorithm – This algorithm defines five classes of interactions that are used by other Feature Tracking algorithms [34]. These interaction classes that are given below and are illustrated in Figure 10:

- **Continuation.** An object continues from time $t_i$ to $t_{i+1}$ with possible rotation, translation, or deformation. Its size may remain the same, grow or shrink.
- **Creation.** A new object appears.
- **Dissipation.** An Object disappears. (Dissipation generally occurs when regions fall below the specified threshold value.)
- **Splitting (Bifurcation).** An object splits into two or more objects.
- **Merging (Amalgamation).** Two or more objects merge to form a bigger object.

![Figure 10: The Feature Tracking Interactions: Continuation, creation, dissipation, bifurcation and merging. Image source – [17].](image)

An overlapping based Feature Tracking was first developed with Octree datastructure [42]. The octree based algorithm works in two phases:
1. VO-test: The first phase detects the overlaps among features and limits the number of candidates to be matched in second phase. This phase has three steps:

1. Segment the dataset into objects and background and store the nodes for each object in the object list.
2. Merge the object lists and sort in ascending order of node ids.
3. Compare the two sorted lists from \( t_i \) to \( t_{i+1} \) to detect the overlap and store these results in overlap table.

2. Best Matching test: this phase finds the correlation between different features.

```plaintext
Feature Tracking
{
  For two consecutive time steps \( t_i \) and \( t_{i+1} \),
  Extract all the features from the two databases and store each feature in its own octree.
  Construct the octree forests \( \mathcal{F}_i = \bigcup_{o_i} O_i^o \) and \( \mathcal{F}_{i+1} = \bigcup_{o_{i+1}} O_{i+1}^o \).
  Use \( O_i^o \) as a template for matching.
  For each feature \( O_i^o \in \mathcal{F}_i \), merge it into the octree forest \( \mathcal{F}_{i+1} \) to:
    - Identify all the overlapping regions of \( O_i^o \) in \( t_{i+1} \).
    - Store this in a list called Overlap\( O_i^o \).
  For each feature \( O_j^o \in \mathcal{F}_j \):
    - Determine bifurcation and continuation:
      - For all combinations of features in Overlap\( O_i^o \),
        Compute \( O_i^o = \bigcup \text{Overlap}\( O_i^o \)\).
        - If the least difference is below the tolerance,
          Mark \( O_i^o \) as bifurcation into the object and remove them all from the search space.
          Next \( O_j^o \).
        Else:
          Determine Agglomeration:
            - For each remaining feature in \( O_i^o \), merge it into the octree forest \( \mathcal{F}_j \) and test for agglomeration:
              - This is the same as bifurcation with the input
              - Take the remaining \( O_j^o \) in \( t_i \) as bifurcation.
              - Take the remaining \( O_j^o \) in \( t_{i+1} \) as creation.
        End if.
    End for.
  End for.
}
```

Figure 11: The overlap based Feature Tracking algorithm using Octree datastructure.

The pseudocode for this algorithm is shown in Figure 11. Please refer to [3,4,30] for more information. However, this algorithm does not work with unstructured data and errors were noticed while tracking small objects. In [3], a solution was implemented to correct the errors noticed while tracking small objects.
Feature Tracking is the process of tracking or following features over a period of time, it is useful for understanding the evolving phenomenon. The interaction between the objects in two consecutive frames is recorded in a file called trakTable. The trakTable file is created from the second frame of the dataset and with each new frame, the information is added at the end of the file. The format of the trakTable file is given in Appendix – II. Each entry in the trakTable signifies the interaction of an object in the frame according to the interaction classes discussed earlier in the section. Some sample trakTable entries are given below:

3 4 -1 5 indicates Merging or Amalgamation.
1 -1 2 3 indicates Bifurcation
6 -1 6 indicates Continuation
-1 7 indicates creation and
2 -1 indicates dissipation

“-1” acts as a delimiter, the numbers to the left of “-1” indicate objects from the previous frame and the numbers to the right of delimiter “-1” indicate the objects from the next frame. If the object dissipates in the next frame, then there is nothing to the right of “-1” and if there is no number to the left of “-1” it indicates the objects are created in next frame.

3.1.2 Enhancements to Feature Tracking algorithm

The original overlap based Feature Tracking algorithm used an octree datastructure [42] that was unsuitable for unstructured or very large datasets. A linked list data structure based solution was developed for tracking features in unstructured grid datasets [4]. This algorithm can be extended to multiblock, multiresolution and adaptive grid structures.
The features are extracted using a region-growing algorithm [35,7] that generates an object list. Each node in object list consists of object id, attributes and all the nodes for that particular object. Merging all the features of a frame and sorting them according to node ids generates a sorted node list. The sorted node list for two frames is compared to detect an overlap. Then the best matching is performed on the overlap table to determine the class of interaction for each object in the frame.

3.1.2.1 Distributed Feature Tracking for processing huge datasets – The Feature Tracking algorithm was unable to handle large datasets. To be able to extract, track and visualize features in large datasets, distributed Feature Tracking capability was introduced in the existing overlap based Feature Tracking algorithm. In this implementation the features were merged using a complete-merge [36] strategy. With this strategy, processors communicate amongst themselves, employ a binary swap algorithm [37] and operate sequentially to merge the features and get tracking results. A partial-merge strategy was also proposed where processors communicate with their neighbors to determine the local connectivity.

3.1.2.2 Feature Tracking for Adaptive Mesh Refinement (AMR) datasets – Chen et. al [5] describes a distributed Feature Extraction and Tracking process for AMR datasets. In AMR datasets, grid points with varying resolutions and features span across multiple grid levels and processors. So, the tracking must be performed across time, across levels and across processors. In this algorithm, the tracking results are first computed temporally across the lowest grid level and then computed across the spatial levels of refinement.
Features formed in higher level are tracked in subsequent time step as Feature Trees. Please refer to [5] for more information on AMR Feature Tracking.

3.2 Applications of Vizlab’s Feature Tracking Algorithms

Feature tracking can be applied to any time-varying 3D dataset. At Vizlab, Feature Tracking was applied to many real-life engineering applications. The structure of the data in these datasets was varied. (Structured Mesh, Unstructured Mesh, etc. as in appendix - II). Some of the application areas are mentioned below:

1) **Autoignition datasets** - Basic feature tracking algorithms can be useful as an analysis tools for combustion datasets by application to a dataset modeling autoignition [28]. In [28], Features defined as areas of high intermediate concentrations were examined to explore the initial phases in the autoigniton process.

2) **Turbulence flows** – Feature tracking can be useful in identifying and temporally tracking hair pin packets and their wall signatures in direct numerical simulation data of turbulent boundary layers [28]. In this work visualization algorithms are validated against the statistical analysis. And they demonstrate that the average geometric packet is representative of strong statistical ones. Also, they presented for the canonical case of an isolated hair pin packet convecting in channel flow, and for fully turbulent boundary layers.

3) **Meteorology** – Feature tracking was applied to the cloud water simulation (Figure 12). This simulation consisted of 25 datasets at a resolution of 35*41*23 [4]. Features were extracted from this dataset and tracking information provided visual cues on object evolution.
Figure 12: The Feature Tracking on cloud water simulation. The top four images are four steps from the simulation. The middle image is one set of features extracted. The bottom graph contains the quantifications of the large object. (Image source – [4]).

4) **Isotropic Turbulent Decay Simulation** – Feature Tracking was applied to LES simulation of the decay of istrophic turbulence in a box in a compressible flow using unstructured tetrahedral. This simulation dataset having 500 frames (or timesteps) showed that the number of objects changes with as the isotrophic turbulence decays (Figure 13) [4].
3.3 Software Implementations of Feature Tracking Algorithms

There are a number of implementations of Feature Tracking algorithms for various visualization software packages. These implementations are discussed below:

3.3.1 AVS/Express implementation

The Ostrk2.0 package is a stand-alone feature tracking software developed in C/C++ on the AVS/Express 6.2 platform by Vizlab [17]. The code can be downloaded from http://vizlab.rutgers.edu/FeatureTrackingCode/. This software package is implementation of Overlap based Feature Tracking algorithm using linked list and works with any
datasets (structured, rectilinear, unstructured grids). The main features of this software package are summarized below:

- **Feature Extraction** - The input dataset is segmented into its features as threshold (specified by the user) connected components. The user choose a percentage threshold and the actual value for this percentage threshold is:
  
  \[
  \text{actual thresh} = p \times (\text{max node value} - \text{min node value})/100
  \]
  
  where \( p \) is the percentage threshold selected. This calculation is performed on a per frame (timestep) basis.

- **Feature Tracking** - The life-cycle of all extracted features is tracked over the number of time-steps of the dataset specified recording all ’events’ that may occur during an objects’ life-cycle specifically merging, splitting, continuation, dissolution or creation.

- **Enhanced Surface Animation** - Users can view an iso-surface visualization of all time-steps with color-coding added to highlight feature events. For example, suppose a feature A in time step 1 splits into features B and C in time-step 2. Then both features B and C will receive the same color as A.

- **Surface Isolation Animation** - The interface in Ostrk2.0 allows you to select a particular feature from the surface animation window (last time-step only) and view its evolution separately in a different window.

- **Attribute analysis and Printing** - The software computes various attributes like volume, mass, centroid, etc., that can be printed on the screen by picking a particular object from a time-step in the enhanced surface animation window.
• **Graph Plotting** - The interface also has a window where the user can view how some frame attributes like number of objects, etc., vary over the time (duration of the tracking).

• **Storing of Feature tracking results** - All attributes for individual objects as well as for all time-steps are stored in files in a pre-defined directory under the users’ run path. The files also include a record of the events, which occur in the life-cycle of an object, e.g., splitting or dissipation.

Currently at Vizlab, the license for AVS software package has expired and has not been renewed further. Feature Tracking cannot be used in isolation with AVS and as AVS needs a valid license, Feature Tracking implementation in AVS, is unavailable. This also means that no modifications or testing can be done on the existing code at Vizlab. However, these plugins can be installed, used and modified by anyone having a valid AVS license.

### 3.3.2 Distributed Feature Tracking Implementation

The distributed Feature Tracking was implemented as a standalone application. In [38], implementation details of the Distributed Feature Tracking process and the code are given. The code for distributed Feature Tracking can be downloaded from [http://vizlab.rutgers.edu/FeatureTrackingCode/](http://vizlab.rutgers.edu/FeatureTrackingCode/). For a large dataset, the code works in parallel mode (distribute the task among a group of processors) to extract features from all the frames of the time-varying dataset and store the tracking information in a .trakTable file. The code is organized in 4 separate directories [38]. These directories are:
**objseg directory** – During the Feature Extraction step each processor loads a local portion of the dataset. At the end of this step each processor generates a .poly file, .oct (object attributes) file and a .trak file. Also, .table file (local object table) is generated on each processor. The code in this directory is compiled using a MPI compile script.

**finalmerge directory** - This part of code includes methods to read in the .table files generated during extraction step to generate a global object table. Also, .poly and .oct files are updated in this step.

**ftrack directory** – here Distributed Tracking is implemented by partial-merge strategy. Again MPI compile script is used to compile the code in this directory.

**score directory** – best match is calculated and features across the frames are correlated.

Each of these directories has to be compiled separately to get the final results. The order of compilation should not be changed, as the output of a particular step is needed by another step. Since, the process of compiling these smaller programs manually was becoming cumbersome, a perl script was written to automate this process of Distributed Feature Tracking. More information about the script, its implementation and the algorithm is given in [38]. This standalone application can be downloaded and used by anyone any machine having MPI installed on it. There is no need to have a cluster to test the code as the perl script was so modified to be able to run on a single processor machine too. Currently, this code is inaccessible to users outside of Vizlab.

### 3.3.3 VisIt Implementation

[23] describes the porting of “Feature Tracking of Time-Varying Scalar Datasets in VisIt”. VisIt that is available free of cost was chosen as alternative to AVS. There are
many other open-source visualization softwares, but VisIt was chosen because of its rich feature set, good architecture and relatively better development environment. [23] describes an implementation to extract the features from a dataset and track those features in VisIt environment. Similar to the AVS implementation, information about the extracted features was written to .poly, .attr, .uocd and .trak files, while the tracking information was written to .trakTable file. This task of extracting and tracking features was done by ‘FeatureTrack’ plugin [23]. Other VisIt users can install these custom plugins by following installation procedure described in Appendix – I. Another group of plugins were designed for visualizing the features extracted from the time-varying datasets. However, these plugins were found to be incomplete and incompatible with newer versions of VisIt. In previous work, these plugins were developed on VisIt version 1.9.0 [23].

In this thesis, we improve the visualization group plugins and add more functionality to it. We retain the plugin names from previous work. We modify some methods in ‘FeatureTrack’ plugin that extracts and tracks features. We re-implement the visualization plugins to develop a flexible, easy to use and scalable framework. We use the VisIt version 1.12.0 and properly document the design decisions for developing visualization group plugins. If these plugins are found to be incompatible on newer versions of VisIt, users can easily generate new plugins by following the methodology discussed here. Chapter 5 discusses all these implementations and modifications while in next chapter we see the VisIt environment.
Chapter 4

VisIt Environment

The basic design of VisIt can be thought of as a client-server model [14]. The client-server aspect allows for effective visualization in a remote setting. VisIt’s architecture allows for parallelization of the server (task of one processor is shared by a group of processors) thereby processing large datasets quickly and interactively. VisIt has been used to visualize many large data sets, including a two hundred and sixteen billion data point structured grid, a one billion point particle simulation, and curvilinear, unstructured, and AMR meshes with hundreds of millions to billions of elements [11]. VisIt follows a data flow network paradigm where interoperable modules are connected to perform custom analysis. The modules come from VisIt's five primary user interface abstractions and there are many examples of each. In VisIt, there are:

- twenty-one "plots" (ways to render data),
- forty-two "operators" (ways to manipulate data)
- eighty-five file format readers, over fifty "queries" (ways to extract quantitative information)
- over one hundred "expressions" (ways to create derived quantities).

Further, a plugin capability allows for dynamic incorporation of new plot, operator, and database modules. These plugins can be partially code generated, even including automatic generation of Qt and Python user interfaces.
4.1 High level design of VisIt

VisIt is composed of multiple separate processes that are sometimes called as components [19]. These components are listed in Table 1:

<table>
<thead>
<tr>
<th>Name</th>
<th>Overview</th>
</tr>
</thead>
<tbody>
<tr>
<td>Viewer</td>
<td>Two primary purposes. First, it centralizes all of VisIt's state. When the state changes, it notifies the other components of the state changes. Second, the viewer is responsible for managing visualization windows, which often includes doing rendering in those windows.</td>
</tr>
<tr>
<td>Gui</td>
<td>Provides a graphical user interface to control VisIt.</td>
</tr>
<tr>
<td>Cli</td>
<td>Provides a command line user interface to control VisIt.</td>
</tr>
<tr>
<td>Vcl</td>
<td>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.</td>
</tr>
<tr>
<td>Mdserve</td>
<td>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.</td>
</tr>
<tr>
<td>Engine</td>
<td>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). The engine sometimes performs rendering, although it is also performed on the viewer.</td>
</tr>
</tbody>
</table>

[Table 1: VisIt's multiple separate processes.  
Table Source - http://visitusers.org/index.php?title=High_level_design]

4.2 Workflow of VisIt

The connections between the various components are shown in the Figure 14. At the lowest level, the communication is done with sockets. However, two separate layers are built on top of that.
The first is for exporting state. The viewer keeps all of its state in various instances of VisIt's \texttt{Attribute\texttt{Subject}} class. UI modules (such as the GUI and CLI) subscribe to this state (refer to [25] by Gamma et al. for more details). Thus, when state changes on the viewer, the \texttt{Subjects} automatically push this state out to its subscribers.

The second is for remote procedure calls (RPCs) [15]. When a component wants another component to perform an action, it issues an RPC.

- The RPCs come via a proxy class [16]. For example, there is a module named `ViewerProxy`. Both the GUI and CLI link in `ViewerProxy` and make method calls to this class. Each method call becomes an RPC.

- Examples of RPCs are:
  - GUI or CLI initiating state change in the viewer
‘Viewer’ causing the ‘Mdserver’ to perform an action, such as opening a file.

‘Viewer’ causing the ‘Engine’ to perform an action, such as drawing a plot.

Consider a scenario where a user performs the following actions:

1) Load a data file
2) Choose a plot (say contour plot)
3) Choose a operator (say slice operator)
4) Click on Draw.

As a result of the above actions, an AVT network (Figure 7) is generated automatically by the VisIt. We briefly discussed this network in Chapter 2. Here, we see the network in detail. Each user action corresponds to building some part of the network. VisIt does not do any processing or visualization until the network is executed. Until that time the ‘Viewer’ just sits with the information. We will see the methods corresponding to the above actions.

1) Load Data File

Figure 15, gives a flow chart of the process that is initiated when user loads a data file. As user opens a file, ‘Mdserver’ component is called to open an ‘avtFileFormat’ and get metadata information from the file. This metadata information is about the data like the type of mesh, scalar variables etc. This part of network building action causes the following actions:
Figure 15: The flow diagram for the process associated with loading of a data file in VisIt.

**Metadata server actions**

1) First, `MDServerConnection::ReadMetaData` method that is defined and declared in `MDServerConnection.C` and `MDServerConnection.h` is called.
   
   - This method opens a file by calling `MDServerConnection::GetDatabase` method.
   
   - This method uses an ‘`avtDatabaseFactory`’ to instantiate an ‘`avtDatabase`’ object.

   The DB factory iterates over viable plugin types

   1) For each viable plugin type, the file format is instantiated.

   2) `avtDatabase::GetMetaData` method is called. This forces the plugin to do some work to see if the file is really of the format's type.

   1) `GetMetaData` method ultimately calls the following methods:
PopulateDatabaseMetaData

GetCycles

GetTimes

2) No calls will be made to GetMesh and GetVar methods.

- The resulting ‘avtDatabase’ is asked to create metadata for the file. This is a no-op, since the meta-data was read when opening the file and that meta-data was cached.

2) Later on, SIL information of the database is requested. This is done in MDServerConnection::ReadSIL method by calling avtDatabase::GetSIL method. After that the following methods are called:

1) avtDatabase::PopulateSIL.

2) avtSILGenerator to populate the SIL entirely from the meta-data.

**Engine actions**

The first of the engine actions it to call the method RPCExecutor<OpenDatabaseRPC>::Execute defined in ‘Executors.h’.

1) The appropriate plugin type is known (from the ‘Mdserver’ which is open) and it is loaded.

2) This calls NetworkManager::GetDBFromCache that does the following:

1) opens the file using the database factory.

2) calls avtDatabase::GetMetaData method.

3) calls avtDatabase::GetSIL method.

4) register the database with the load balancer.
2) **Adding a plot or operator** - as the user chooses the plots and adds operators, the engine responds by constructing an AVT network. There is no communication between different components of VisIt. The viewer just sits with the information and does nothing with it.

3) **Clicking on draw** - As the user clicks on ‘Draw’, the AVT network is executed and the following steps are performed in this process.

- **Preparing for scalable rendering**
  - In non-scalable rendering, the resulting surface is transferred to the ‘Viewer’ and rendered locally. The rendering is done using an ‘avtPlot’s’ ‘mapper’ module that is called in the context of a VisWindow's visualization window.
  - In scalable rendering, the surface is rendered in parallel, and the ‘Engine’ transfers an image back to the ‘Viewer’.

- **Stating which file to use as the source**

  `RPCExecutor<ReadRPC>::Execute` method from ‘Executors.h’ is called. This method calls `NetworkManager::StartNetwork` method that performs the following actions:
  
  1) The ‘avtDatabase’ is identified (it was already created during an ‘OpenDatabaseRPC’).
  
  2) An ‘avtTerminatingSource’ is gotten from the ‘avtDatabase’.
  
  3) An ‘avtExpressionEvaluatorFilter’ is added to the pipeline (at the top).
  
  4) The ‘avtSILRestriction’ is registered.

- **Setting up the operators**
1) \texttt{RPCExecutor<PrepareOperatorRPC>::Execute} method is called. This must be called first to instantiate the correct type of attributes, so that the subsequent call to “Add Operator” will be able to load the attribute values into this instance.

2) \texttt{RPCExecutor<AddOperatorRPC>::Execute} is called. This method calls \texttt{NetworkManager::AddFilter} method. This method ensures that:

   1) The proper plugin type is loaded.
   2) An ‘\texttt{avtFilter}’ is instantiated and registered with a "workingNet".
   3) The attributes of the filter are set.

- Setting up the plots

  This is similar to the setting up of operators. The following actions take place:

  1) \texttt{RPCExecutor<PreparePlotRPC>::Execute} method is called. This must be called first to instantiate the correct type of attributes, so that the subsequent call to "MakePlot" will be able to load the attribute values into this instance.

  2) \texttt{RPCExecutor<MakePlotRPC>::Execute} method is called. Then,

  1) \texttt{NetworkManager::MakePlot} method is called.

     1) The proper plugin type is loaded.

     2) An ‘\texttt{avtPlot}’ is instantiated and registered with a "workingNet".

     3) The attributes of the plot are set.

     2) An Id is obtained from the network manager and returned to the viewer. This Id is used to refer to this plot in the future. (For picks, etc.)

- Executing the network

  \texttt{RPCExecute<ReadRPC>::Execute} method is called. This method calls the
following two methods:

1) `NetworkManager::GetOutput`. This method sees that:

1) Each module of the pipeline is connected.

2) `DataNetwork::GetWriter` method is called, which in turn calls `avtPlot::Execute` method. This method returns either a geometry, or a NULL object saying that we need to kick into Scalable Rendering mode.

2) The output is sent through a socket with a `WriteData` call.
   - The output comes as an `avtDataObjectWriter` instance.
   - This output may be either a data set or a NULL object, indicating that we should switch to Scalable Rendering mode.

4) Subsequent actions, like queries and picks, cause the engine to connect new sinks to that AVT network.

### 4.3 Plugin types in VisIt

VisIt supports development of custom plugins. In VisIt, plugins are divided into three categories: plots, operators and database readers and writers.

**Plot**

A plot is a viewable object, created from a database that can be displayed in a visualization window. VisIt provides several standard plot types that allow you to visualize data in different ways. The standard plots perform basic visualization operations like contouring, pseudocoloring as well as more sophisticated operations like volume
rendering. All of VisIt’s plots are plugins so you can add new plot types by writing your own plot plugins.

**Operator**

An operator can be considered as a filter applied to a database variable before the compute engine uses that variable to generate a plot. VisIt provides several standard operator types that allow various operations to be performed on plot data. The standard operators perform data restriction operations like planar slicing, spherical slicing, and thresholding, as well as more sophisticated operations like peeling off mesh layers. All of VisIt’s operators are plugins and you can write your own operator plugins to extend VisIt in new ways.

**Database**

VisIt can create visualizations from databases that are stored in many types of underlying file formats. VisIt has a database reader for each supported file format and the database reader is a plugin that reads the data from the input file and imports it into VisIt [22]. If VisIt does not support your data format then you can first translate your data into a format that VisIt can read (e.g. Silo, VTK, etc.) or you can create a new database reader plugin for VisIt.

**4.4 Adding new plugins in VisIt**

VisIt comes with a graphical plugin creation tool that greatly simplifies the process of creating new plugins. The user describes the properties of the plugin and then the tool generates most of the code necessary to implement the plugin. The only code you need to write is the C++ code that actually performs the operation.
Steps for creating a new plugin

VisIt provides some tools to help developers in adding any new functionality. These tools can be found under the following location:

<visithomefolder>/src/bin>

More information about these tools can be found in VisIt’s User Manual [20]. If this folder has been added to the system path, then the tools in this folder can be accessed from any location, by just typing the name of the tool. Otherwise the full path must be specified. For example, if we want to use xml2edit tool in some folder e.g., /home/admin. Then the commands to run this tool are:

cd /home/admin
xmledit
<visithomefolder/src/bin>xmledit

For the rest of commands in this thesis, we assume that VisIt’s bin folder is added to system path. To create new plugins using these tools, one must follow the following steps:

1) Create a directory by the plugin name. The location of this folder depends on the type of the plugin. If the plugin is operator, it should go under <visithomefolder/src/operators>, if it’s a plot it should be in <visithomefolder/src/plots> and if it’s a database plugin then it should be at <visithomefolder/src/database>

2) Change to the location of the directory and run xmledit by following commands:

cd < visithomefolder/src/ <plugintype>/ <pluginname> >
xmledit
A window similar to Figure 16 would appear on the screen. An untitled xml file appears on the screen and user is required to fill in the information pertaining to the plugin. The information includes the name of the plugin, the type of the plugin, attributes and so on (refer to VisIt Developer’s Manual for more details). Attributes are the parameters that allow users to interact with the plugin. For example, opacity is an attribute and can be changed by the users via a slider. A plugin can have one or more than one attributes. After the attributes are selected, their description entered and all the information provided, the file is saved as an xml file by the same name as that of the plugin. So, if the plugin is
TrackPoly, then the xml file should be saved as TrackPoly.xml in TrackPoly folder.

3) Run xml2plugin. This will automatically create a framework to work on. This can be done as follows:

```
cd cd <visithomefolder/src/ <plugintype>/ <pluginname> >
xml2plugin <pluginname>.xml
```

4) The framework depends on the type of plugin. The files generated by VisIt depend on the type of the plugin. Table 1 gives a list of files generated for Feature Tracking plugins. These plugins have to be compiled before using, and some of the methods in these files have to be edited to compile. Again, the methods to be edited/modified depend on type of plugin.

While adding a plot, the important methods are:

```c
virtual void SetAtts (const AttributeGroup*);
virtual avtMapper *GetMapper (void);
virtual avtDataObject_p ApplyOperators (avtDataObject_p);
virtual avtDataObject_p ApplyRenderingTransformation (avtDataObject_p);
virtual void CustomizeBehavior (void);
virtual void CustomizeMapper (avtDataObjectInformation &);
```

All these methods part of the class avt<plot-name> and hence would be defined and declared in avt<plot-name>.C and avt<plot-name>.h. Depending on the aim of the plugin, different methods have to be modified.

While adding an operator, the important method is:
vtkDataset*    **ExecuteData** (vtkDataSet, int, std::string)

This method is part of class `avt<operator-name>Filter`, is declared and defined in `avt<plot-name>.h` and `avt<plot-name>.C`. Any processing or filtering on the dataset can be introduced by adding some lines of code in this method.

While **writing a database writer**, there are four basic methods, which must be implemented, although any of these methods can be no-ops (i.e. you can leave them empty with just `{;}`). The specifics on how these methods are called is mentioned in `/src/avt/Pipeline/Sinks/avtDatabaseWriter.C` and the signature of these methods are:

```cpp
void OpenFile (const std::string &, int);
void WriteHeader (const avtDatabaseMetaData *,
                  std::vector<std::string>&, std::vector<std::string>&,
                  std::vector<std::string> &);
void WriteChunk (vtkDataSet *, int);
void CloseFile (void);
```

In case of database reader, the following method needs to be implemented.

```
vtkDataSet* avtPolyFileFormat::GetMesh(const char*);
```

This method is described in `avt<Databasename>FileFormat` class and changes in here would accomplish the task.

5) Then Compile and run!
Chapter 5

Feature Tracking & Visualization in VisIt

5.1 Motivation

Vizlab has pioneered the use of the Feature Tracking for analyzing the time-varying datasets. Over the years, the Feature Tracking algorithm has evolved to incorporate new functionalities (as discussed in Chapter 3). The overlap based Feature Tracking algorithm using a linked list datastructure was implemented as a module in AVS. However, open-source solutions were also explored. Among the open-source visualization software packages, VisIt was chosen to port the Feature Tracking capability. VisIt has a rich feature set, supports distributed and parallel processing for large datasets and has a robust architecture that makes it a good choice. Extending new capabilities like the Feature Tracking & Visualization in VisIt would add a lot more value to VisIt and benefit the users tremendously. Previous work [23] in porting the Feature Tracking into VisIt was successful in extracting and tracking the features from a time-varying dataset. But, the visualization plugins were found to be incomplete and incompatible with the newer version of VisIt. Hence this thesis was written to:

- Successfully implement the “Feature Tracking & Visualization” capability in VisIt.
- Add intuitive and user-friendly functionalities like ‘Selective Feature Tracking’ and ‘Picking Object by Mouse clicks’.
- Design a flexible, intuitive, easy to use and solid framework that lays a strong foundation for the next generation Feature Tracking Algorithm.
- Document the design decisions and the workflow of Visit so that it could be useful to the users, who wish to add new capabilities in VisIt.

- Organize all the existing Feature Tracking publications, implementations, code and datasets for the benefit of the Vizlab users and researchers.

There is also a technical reason to support VisIt for implementing the Feature Tracking & Visualization. The architecture of VisIt is such that it aids in separation of the two main modules for the Feature Tracking & Visualization process. These modules are:

- The Feature Extraction and Tracking module and

- The Visualization module.

First, the features are extracted one by one from all the frames. While extracting these features, the .trakTable file is updated starting from the second frame. After updating the .trakTable file for all the frames, the results are rendered on the screen. If a scientist is analyzing some experimental simulation dataset, and after visualizing the few initial frames scientist might find some interesting phenomenon. The scientist decides to skip the visualization of those initial frames and visualize the features from the time interesting phenomenon occurs. As this change in visualization criteria does not change in the input data, the feature extraction and tracking process should not be repeated. If there is no separation between the modules extracting the features and visualizing the results, the process of extraction is repeated whenever the scientist varies the visualization criteria. For large datasets, this repetition increases the program execution time. When the modules for the feature extraction and visualization are separate, the feature extraction and tracking is performed only once. After that the results can be visualized in any possible manner without extracting the features again. As the features
extraction process is not repeated again and again, the visualization process becomes faster and more efficient.

5.2 Feature Tracking & Visualization Functionalities

The features are extracted from each frame of a time-varying dataset and the results are visualized on the screen. The visualization results from the individual frames when stacked together and played as an animation, provides a greater insight about the evolving features in the time-varying datasets. Assigning colors according to the tracking information would be helpful in visually tracking the objects across the frames. As the amount of data rendered on the screen increases, tracking and detecting events becomes challenging. It would be useful, if users were able to select the objects of interest and track only those objects. In this way, only few objects are rendered on the screen. For tracking only the features of interest, users should have an option of either specifying the object numbers as a list or selecting the interesting features with mouse clicks. Our implementation provides these functionalities and we discuss about each of them in detail in the following sections.

5.2.1 TrakTable based Color-coding

From Figure 17, it is clear that the objects in each frame on the left side of the figure have colors assigned to them arbitrarily. There is no correlation between the objects and the colors assigned to them. In time-varying datasets, an object from one frame can move around in the next few frames of the dataset. The objects can change their shapes, split into smaller objects, form a bigger object by merging or continue as before. Since the
objects continue to exist (although in a different form) in subsequent frames they should get the same color as in first frame. If a new object is formed in a particular frame, then it should be assigned colors that do not correspond to any of the existing objects. A coloring scheme based on the trakTable that maintains the tracking information, would produce results similar to those on the right side of Figure 17.

Figure 17: The tractable based coloring. The images on the left half show random colors being assigned to objects in the frame, on the right shows the results of feature tracking with proper color codes. The figure shows 3 consecutive frames of vorts data. Lets study the evolution of object A and Object B in (a), (b) and (c). The necessity for assigned coloring based on track tables becomes evident in (c), as its clear the object A has split into two and both the parts retaining the same color.
When the objects follow the tracker-based coloring scheme, it becomes very easy for the users to keep track of these objects visually. In Figure 17, we can easily identify that the Object A continues in 2 frames and splits in the third frame, while the Object B moves in all the frames of the dataset with a very little change in its size and the location. This behavior can be useful in the process of analysis and hypothesis building.

5.2.2 Selective Feature Tracking

Sometimes, we just want to visualize and track only few objects that are of interest to us. We should have a mechanism to specify a list of interesting objects and this list when passed to the program renders only those objects mentioned in the list. We can either remove the other objects or employ special techniques to give such an illusion. One way of achieving this illusion can be to adjust the transparency values to make the objects almost transparent (very low percentage opacity) and reduce the visual clutter. The users should have the ability to control the opacity value for both the object to be tracked and the objects that are to be made transparent. For implementing this functionality, we assume that the user knows the object numbers of all the objects in a frame and selects the appropriate object by writing those numbers in a list. This assumption becomes unviable as the number of objects in a frame increases. To assist the users in such cases, we modify our implementation to let the user select objects at the click of a mouse.

5.2.3 Picking Objects by Mouse Click

When presented with a lot of data on the screen, it is intuitive for the user to click on a particular region on the screen and get more information about that region. VisIt extends this concept by providing more information about the objects to the users. When a user clicks on any objects in the frame in either the ‘Node Pick’ or ‘Zone Pick’ mode, a ‘Pick’
window pops up that provides additional information about the object. We extend this facility in VisIt to our advantage and build a list of objects for Selective Feature Tracking with mouse clicks. The users can clicks on all the objects of interest in a frame and the plugin would automatically extract the node number of those objects and build a list. The users also have an option of editing the list manually. Once the user finishes this operation and clicks on ‘Apply’, the objects mentioned in the list are rendered on the screen and the rest of the objects are made transparent.

### 5.3 Custom Plugins

To extend the functionalities as mentioned in the previous section, we need to develop custom plugins in VisIt. In [23], the Feature Tracking was ported to VisIt environment; four plugins were developed for this purpose.

Figure 18: The Feature Tracking Plugin development in VisIt.

Figure 18 describes the four plugins that were taken from the previous work. These plugins are categorized into two groups. The first group is responsible for the Feature Extraction & Tracking. This group has a plugin named ‘FeatureTrack’ that is an operator. The second group has three plugin; ‘Poly’ a database reader, ‘TrackPoly’ an
operator and ‘PolyData’ a plot for visualizing the results. The descriptions of these
plugins are as follows:

5.3.1 The Feature Extraction & Tracking group plugins

The plugin in this group is:

- **FeatureTrack** - this operator extracts the meaningful features from the raw data
  and tracks them. In [23], this module generated .poly, .attr, .uocd, .trak and
  .TrakTable files. In addition to those files, we generate one more file per frame in
  this module. This file has the colormap information for objects in the frame (refer
  Appendix – II for the format of this file).

5.3.2 The Visualization group plugins

The plugins in this group are:

- **Poly** - this database plugin reads the information in .poly files, gathers the
  information metadata about the dataset and creates a ‘vtkDataset’ object to be
  passed on to the network. In [23], this plugin generates a ‘curpoly.txt’ file (refer
  Appendix II for the file format) to keep a track of the current frame. In this thesis,
  we modify the format of the curpoly.txt file from the previous work.

- **TrackPoly** - this operator plugin reads the ‘curpoly.txt’ file and replaces the last
  line in the file (for indicating the frame number) with the current frame number.

- **PolyData** - a plot should be able to visualize (or display) the results on the screen.
  The ‘PolyData’ plot aims to visualize the objects listed in a .poly file by
  displaying them on the screen. The ‘PolyData’ plot employs the trakTable based
  coloring scheme to differentiate objects in a frame and track them over a period of
  time. The ‘Pseudocolor’ plot assigns colors based on the iso-surface values and
allocates distinct colors to the distinct iso-surfaces. If there are three different iso-surfaces, then three distinct colors like red, blue and green are assigned to them.

Now in the next frame, if one of the objects splits into two smaller objects, then there are a total of four objects or iso-surfaces in the next frame. All these four distinct iso-surfaces (or objects) get different colors. To assist the users in tracking objects visually, objects formed by bifurcation of a larger object should get the same color as the parent object. As the ‘Pseudocolor’ plot does not assign colors based on the interactions amongst the objects in successive frames, it is unsuitable for visualization of time-varying data for analysis and hypothesis formation. Other built-in plots are also not suitable for visualizing the time-varying data; hence we had to design a new plot for this purpose.

5.3.3 Auto-Generated Files

VisIt provides the developers with some tools to help create new plugins. These tools generate a folder for the plugin with all the essential files needed by VisIt in it. Table 2, lists all auto-generated files during the creation of all the four Feature Tracking plugins using xml2plugin tool. Since, the ‘FeatureTrak’ operator was modified from the previous work [23] without any change to the operator attributes (specified in .xml file), the ‘FeatureTrack.xml’ file was copied as it is from the previous work. The steps involved in creating the plugin framework are listed below:

- Use xml2plugin tool (under /src/bin/ directory) to generate the plugin framework.

The commands for using xml2plugin are as follows:

```
cd <visithomedirectory/src/operator/FeatureTrack>
../../bin/xml2plugin FeatureTrack.xml
```
In case VisIt’s bin folder is added to your system path, then just type:

```
xml2plugin  FeatureTrack.xml
```

<table>
<thead>
<tr>
<th>Auto Generated Files</th>
<th>FeatureTrack</th>
<th>TrackPoly</th>
<th>PolyData</th>
<th>Poly</th>
</tr>
</thead>
<tbody>
<tr>
<td>avt&lt;PluginName&gt;Filter (.C , .h)</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td></td>
</tr>
<tr>
<td>avt&lt;PluginName&gt; (.C , .h)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>avt&lt;PluginName&gt;FileFormat (.C , .h)</td>
<td></td>
<td></td>
<td></td>
<td>√</td>
</tr>
<tr>
<td>avt&lt;PluginName&gt;Options (.C , .h)</td>
<td></td>
<td></td>
<td></td>
<td>√</td>
</tr>
<tr>
<td>avt&lt;PluginName&gt;Write (.C , .h)</td>
<td></td>
<td></td>
<td></td>
<td>√</td>
</tr>
<tr>
<td>Input (.C , .h , .java)</td>
<td>√</td>
<td>√</td>
<td></td>
<td></td>
</tr>
<tr>
<td>pyInput (.C , .h)</td>
<td>√</td>
<td>√</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;PluginName&gt;Attribute (.C , .h , .java)</td>
<td></td>
<td></td>
<td></td>
<td>√</td>
</tr>
<tr>
<td>py&lt;PluginName&gt;Attribute (.C , .h)</td>
<td></td>
<td></td>
<td></td>
<td>√</td>
</tr>
<tr>
<td>Qvis&lt;PluginName&gt;Window (.C , .h)</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td></td>
</tr>
<tr>
<td>Qvis&lt;PluginName&gt;Window_moc (.C , .h)</td>
<td>√</td>
<td>√</td>
<td></td>
<td>√</td>
</tr>
<tr>
<td>&lt;PluginName&gt;CommonPluginInfo (.C , .h)</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
</tr>
<tr>
<td>&lt;PluginName&gt;EnginePluginInfo (.C , .h)</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
</tr>
<tr>
<td>&lt;PluginName&gt;PluginInfo (.C , .h)</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
</tr>
<tr>
<td>&lt;PluginName&gt;GUIPluginInfo (.C , .h)</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
</tr>
<tr>
<td>&lt;PluginName&gt;MDServerPluginInfo (.C , .h)</td>
<td></td>
<td></td>
<td></td>
<td>√</td>
</tr>
<tr>
<td>&lt;PluginName&gt;ScriptingPluginInfo (.C , .h)</td>
<td>√</td>
<td>√</td>
<td></td>
<td>√</td>
</tr>
<tr>
<td>&lt;PluginName&gt;ViewerPluginInfo (.C , .h)</td>
<td>√</td>
<td>√</td>
<td></td>
<td>√</td>
</tr>
</tbody>
</table>

Table 2: List of auto-generated files in VisIt.
• Copy the files as mentioned in Table 2, in this folder. To make your operator function properly, you need to modify some of the auto-generated files. Since, we are using most of the work for this operator from previous work [23], instead of modifying the auto-generated files from beginning, we just use the files from previous work.

• Now, compile the ‘FeatureTrack’ code and the ‘FeatureTrack’ operator plugin is ready to be used. The following commands are used to compile an operator.

```
  cd <visithomedirectory/src/operator/FeatureTrack>
  make
```

**Caution:** Whenever, a plugin has to be installed that is written by someone else and not included in VisIt, the above steps should be followed. Merely, copying all the files into the new folder by the name of the plugin and compiling that plugin won’t work. Since the build environment is different on different machines, it is best to leave this job of figuring out the right build environment to VisIt and generate appropriate makefiles.

### 5.3.4 Newly added files

In addition to modifying the auto-generated files, some new files were also added. Table 3 gives the list of all the files that have been added in the Feature Tracking plugins. Some of the files are marked with an (*), it means that these are auto-generated files and they have been modified to add new methods or change the functionality of the existing method. The new files are added to the `<pluginname>` directory. VisIt is so structured that all the plugins (either existing in VisIt or new ones) are grouped in `<visithomefolder/src>` directory under the categories of plot, operator or databases. So, a file in the ‘FeatureTrack’ operator column is added to
<table>
<thead>
<tr>
<th>FeatureTrack Operator</th>
<th>Polydata Plot</th>
<th>TrackPoly Operator</th>
<th>Poly Database</th>
</tr>
</thead>
<tbody>
<tr>
<td>avtFeatureTrackFilter (.C, .h) *</td>
<td>avtPolyDataPlot (.C, .h) *</td>
<td>avtTrackPolyFilter (.C, .h) *</td>
<td>avtPolyFile Format (.C) *</td>
</tr>
<tr>
<td>Ftrack/FeatureTrack (.C, .h)</td>
<td>QvisPolyDataPlot Window (.C, .h) *</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ftrack/FeatureTrackUtil (.h)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Interface/InterfaceUtil (.h)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ObjSegment/cellinfo (.C, .h)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ObjSegment/nodeinfo (.C, .h)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ObjSegment/objectinfo (.C, .h)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ObjSegment/input (.h)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ObjSegment/ObjectSegment (.C)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ObjSegment/ObjSegmentUtil (.h)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ObjSegment/stobject1 (.h)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ObjSegment/stRGB (.h)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ObjSegment/Util (.h)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 3: List of all files created/modified. * indicates the auto-generated files which were modified.

<visithomefolder/src/operators/FeatureTrack>. The new files added can be either be header files (.h), implementation files (.C) or both. In the table there are files like ‘ObjSegment/stRGB.h’, which means the file name is ‘stRGB.h’ and it’s under a folder called ‘ObjSegment’. For the ‘FeatureTrack’ operator plugin, two new methods have
been added to the ‘avtFeatureTrackFilter’ class (Figure 14). Details of these new methods are given later on in this section. There has been no change to the rest of the files mentioned in table 2 for the ‘FeatureTrack’ operator, hence they are retained as it is from the previous work [23].

5.4 Feature Tracking & Visualization Workflow in VisIt

The Feature Tracking is a two-step process (Figure 19). The first step extracts and tracks the features, while the second step visualizes the features. Step 2 is dependent on step 1 as some of the files (like .poly) are generated in step 1 are an input to the step 2. Step1 has to be executed atleast once to generate all the necessary files required in step 2. Now, step2 can be repeated as many times as the user wants. Step1 creates the files as mentioned in Figure 19 (5 files per frame and one trakTable file for the entire dataset). The database reader loads a file (provided the format is known to VisIt), caches the metadata information about the file and passes on this information to the ‘FeatureTrack’ operator. The metadata information includes the number of nodes, their connectivity, scalar values at these nodes and sometime the value of the normals at these nodes. Now, the ‘FeatureTrack’ operator is ready to segment the data, extract the features and write the information about the extracted features to the files. The ‘Feature Track’ operator needs to have more information about the dataset, like the starting and ending frame numbers, the percentage threshold for segmenting the objects, the location of the files and a few other things (refer to Appendix – I for details about the the ‘FeatureTrack’ operator attributes). These parameters are called the Attributes and they are a mechanism for providing the user-input to the plugins.
Figure 19: The Feature Tracking & Visualization process in Visit. It is a two step process; step 1 does the feature extraction and generates the files for step 2. Step 2 uses the poly file created from step 1 and with the TrackPoly operator and Polydata plot does the visualization and produces an output.

These attribute values are highly dependent on the datasets and the kind of data they represent, so providing a default value does not make sense. The ‘FeatureTrack’ operator
generates all the necessary files and passes on the ‘vtkDataSet’ object to the plot. As the operator accomplishes the main aim of the Feature Extraction and Tracking step, the plot has no useful function. As VisIt does not allow the operator to be applied on the data directly without selecting any plot; a plot had to be chosen as a dummy. We select the ‘Pseudocolor’ plot as a dummy plot to finish step 1. The output on the screen varies from one to two objects or a blank screen. This is fine as along the intermediate file are generated. Once the files are generated from this step, the visualization plugins renders the information in these files on to the screen in step 2. The information about the extracted objects is in the .poly files. The format of the .poly files is unknown to VisIt, hence a ‘Poly’ database reader was written for this purpose [23]. During the loading of the .poly files, the plugin generates a curpoly.txt text file. This file has the information about the file name and number of the objects in the file and the current frame number (which is extracted from file name). Now the ‘vtkDataSet’ object created by reading the .poly files is passed to the ‘TrackPoly’ Operator. The operator does nothing to the ‘vtkDataSet’ object, but it updates the frame number in the ‘curpoly.txt’ file and passes the ‘vtkDataSet’ object to the plot. The plot opens the ‘curpoly.txt’ file, reads the current frame number and fetches the corresponding colormap file for the frame (these files are created in step 1). From the colormap information a ‘vtkLookUpTable’ object is created. The ‘vtkLookUpTable’ is an array of R,G,B and alpha values. The plot constructs the colormap by reading the colormap file and passes the information to the ‘vtkDataSet’ object. After calling the appropriate methods, colormap is applied to the objects and the results are rendered on the screen. The process then repeats for the other files (or frames) in the database.
5.5 Modifications to the Feature Tracking & Visualization Plugins

Figure 20: A Tree list view of methods added to the Feature Tracking & Visualization plugins.

Figure 20 is a tree structure describing all the new methods added for achieving the target Feature Tracking & Visualization functionalities. The methods are grouped according to the functionality and indicate the classes that need to be modified in the respective plugins to accommodate new methods. In the figure, only the names of the methods are
given and not their signatures. These methods and their signature are discussed later on in this section, according to the functionality for which they were added/modified.

5.5.1 ThetrakTable based coloring

From the workflow of the Feature Tracking algorithm (figure 20), the ‘FeatureTrack’ operator seems to be the best place for generation of colormaps and the ‘PolyData’ plot for the application of the colormap to the data. Implementing this functionality meant changes in all the four plugins.

5.5.1.1 New methods in the FeatureTrack Operator (for the colormap generation)

The colormaps are generated by modifying and adding the methods in the ‘avtFeatureTrackFilter’ class (see figure 20) defined and declared in the ‘avtFeatureTrackFilter.C’ and ‘avtFeatureTrackFilter.h’ files. The signature of these methods and their behaviors are given below:

1. int avtFeatureTrackFilter::BeginFeatureTrack (char *, int &);

The C++ equivalent of the following steps was added to this method. These steps are:

1) Get the current frame number

2) If it is the first frame

   • Open a new file called, colormap1.txt

   • Get number of objects in current frame

   • Run a loop from 0 till number of objects in the frame, for each iteration:
     • generate three random integer values (use rand()) and store them as R, G and B values. Let alpha be 1.
     • Write down the all the three R, G, B and alpha to the colormap1.txt file
     • End the loop and close the file.
3) If it’s not the first frame, then open a new colormap file for that frame. If current frame number is 2, then create and open a file called colormap2.txt.

4) Allocate a vector <string> to store the color information for Frame1 (previous frame) and Frame2 (current frame) objects.

5) Open the colormap file for previous frame and fill in Frame1 vector from those values. The number of lines in the colormap file indicates number of objects in previous frame, and that is the size of Frame1 vector. Each entry in the vector have R, G, B and alpha values in it, as one string separated by blank spaces.

6) Open the .trakTable file and fill the TrakTable vector from the entries corresponding to the current frame. Each element of the TrakTable vector is a string. Use the ReadCurrentFrameTrackInfo routine for this purpose.

7) Iterate for the number of entries in the TrakTable vector and for each iteration:

   1. Parse each element of the TrakTable vector, which is a string. For example,

      \[3 \ -1 \  2\] can be one of the entries, -1 is the delimiter, whatever is to the left of -1 is previous frame object and the number on the right represent current frame objects.

   2. Separate out the previous frame and the current frame objects

   3. Set the color for the current frame objects of Frame2 based on the previous frame colors (from Frame1 vector). i.e. for 3 -1  2; object2 of the current frame (Frame2) should get the color of object3 from previous frame (Frame1).

   4. If there are more than one current frame objects, set the same color for all those objects
5. If more than one objects from previous frame combine, use the
find_merge_color routine to get the color of the heaviest object as the
final color.

- If the current frame objects are newly created, then generated another set
  of R,G and B values randomly and set alpha as 1.

8) After step7, Frame2 vector is populated with the colors based on the Frame1
colors, write these colors to the corresponding colormap file and close the file.

2. void avtFeatureTrackFilter::ReadCurrentFrameTrackInfo (string, int,
vector<string>&);

This helper method or routine parses the .TrakTrable file and extracts the information for
the current frame. The format of the .trakTable file is given in Appendix – II. If the
current frame number is 2, this method will extract all the lines in the trakTable file
starting from the “Frame #2” in the file till the next frame i.e “Frame #3”. All the lines
are written to a vector of string named TrakTable, while skipping the blank lines if any.

3. vector<string> avtFeatureTrackFilter::Tokenize (const string &);

This method or routine breaks a string into tokens; in this case delimiter is a blank space.
This is useful on the TrakTable vector, where each line has certain numbers encoded as a
string and this routine breaks those numbers into separate tokens and processes them.

4. string avtFeatureTrackFilter::find_merge_color (Frame &,
vector<string> );

This method or routine finds the mass of the heaviest object amongst the given list and
returns the color corresponding to the heaviest object. One of the inputs for this method is
a frame type dataset that has data members to get the number of objects in a frame and
the mass/volume of each object in the frame.
5.5.1.2 Modifications to the Poly Database Reader

In the ‘Poly’ Database reader, we made changes to the following method:

```
vtkDataSet* avtPolyFileFormat::GetMesh (const char*);
```

The name of the current .poly file is an argument to this method. From this .poly file name, frame number can be extracted. This method reads the .poly file and generates a mesh from the data values in .poly files. In this process, it finds out the number of the objects in the frame. In [23], few lines of code were added to this method to write the file name and number of the objects in a frame to the ‘curpoly.txt’ file. In this thesis, we add an extra line to the ‘curpoly.txt’ file. This extra line represents the frame number that initially starts from 0 and is modified by the ‘TrackPoly’ operator for each frame.

5.5.1.3 Modifications to the TrackPoly Operator

The method modified in the ‘TrackPoly’ operator is from the ‘avtTrackPolyFilter’ class and is defined in the ‘avtTrackPolyFilter.C’ file. The signature of the modified method is:

```
vtkDataSet* avtTrackPolyFilter::ExecuteData (vtkDataSet *, int, std::string);
```

This is an auto-generated method had to be re-implemented for the operator to function properly. In this method, we perform the following actions:

- Get the current path from operator attributes
- Open the ‘curpoly.txt’ file, this file would be at the current path, that user enters as an attribute to the operator.
- Modify the last line of ‘curpoly.txt’ file representing the frame number by adding 1 to the existing value.
- Close the file
Why modify the curpoly.txt file from the TrackPoly operator rather than the Poly reader?

The files are automatically grouped by VisIt into a database that could be played as an animation. Clicking on “Open” or double clicking on the file loads the first file in the database. From there on, other files are loaded by clicking on the animation control buttons, “forward, reverse and play”. The ‘curpoly.txt’ file is generated by the ‘Poly’ plugin. The method generating the ‘curpoly.txt’ file is not called, when the files were loaded from the animation control panel. This is because the ‘curpoly.txt’ file was generated in a method that gets the metadata information of the dataset (avtPolyFileFormat::GetMesh()). Once VisIt reads this metadata information about a dataset, it caches it and this method is not called again. As a result the ‘curpoly.txt’ file is not written or updated when the files are loaded by clicking on the “Play” button in the animation control panel. However, it was noticed that the operator methods are called each time (whether the file is played as part of animation or its loaded by double clicking on it), so the task of updating the curpoly.txt file was moved to the ‘TrackPoly’ operator methods. The ‘Poly’ reader plugin just creates the file and the ‘TrackPoly’ plugin just updates it for every time frame. To double check that the frame number information is correction, another file named ‘CurrentFile.txt’ was generated. This file contains a lot more information about the dataset and can replace some of the attributes that the users need to fill in. Details of this file are given in the section 5.6.
5.5.1.4 New methods in the PolyData plot

For the trakTable based coloring, few methods had to be added or modified in the ‘avtPolydata’ class (figure 20) that is defined and declared in the ‘avtPolydata.C’ and ‘avtPolydata.h’ files. The signature and behavior of these methods are described below:

• void avtPolyDataPlot::SetAtts (const AttributeGroup *a);

After the processing the data by applying the operator on it, VisIt calls this method. Here we can set the attributes for the plot, to display the results. These attributes come from the user. In this method, we define how these attributes are to be interpreted. In the ‘PolyData’ plot, GetInfoAboutThisFile routine provides the information about the current file being handled by the plot. After that the method to find out the current polyfile is called, and then the setColorTable routine is called that sets the colors for the objects in a frame.

2) void avtPolyDataPlot::GetInfoAboutThisFile ();

This routines helps in finding the information about the .poly files, like the starting and ending frame numbers, the number of steps in the dataset and the dataset name. This information is useful for verifying the current .poly file that we are working on.

3) vtkLookUpTable* avtPolyDataPlot::SetColorTable (string);

This routine parses the colormap file, reads the R, G, B and alpha values into different vectors of doubles. After that the routine creates a ‘vtkLookUpTable’ object and passes the R, G, B and alpha information to an instance of the ‘vtkLookUpObject’ object, builds the look-up table and applies it to the ‘vtkDataSetObject’ object.
5.5.2 Selective Feature Tracking

Selective Feature Tracking lets a user select and track only the features of interest amongst all the extracted objects in a frame (or timestep). As can be seen in Figure 21, only one object is visible i.e., only one object is selected and the rest of the objects are made transparent. Selective Feature Tracking lets the users concentrate on the objects of interest and reduce the visual clutter.

![Figure 21: Selective Feature Tracking on the Vorts dataset. Here, we choose to track only one object that is of interest to us. The rest of the objects in the frame are made transparent that reduces the visual clutter.](image)

A list of the objects to be tracked is provided as an input to the program. As the list is an input, it should be one of the attributes of the ‘PolyData’ plot. The user can control the opacity of the objects that are being tracked. An opacity slider is provided for this purpose. Another slider is provided to control the opacity of the rest of the objects in the frame. This slider has a default value of 5% that can be changed. When the user clicks on ‘Apply’, a list of non-opaque objects to be tracked in subsequent frames is generated. This list of non-opaque objects depends on the initial list provided by the user and the trakTable file. Any processing that has to be done after the user clicks on ‘Apply’ and
before the results are rendered on the screen can be achieved by modifying the methods in the ‘QvisPolyDataPlotWindow’ class. To generate a list of non-opaque objects, new methods are added to the ‘QvisPolyDataPlotWindow’ class (Figure 20). To generate the list of non-opaque objects for rest of the frames, the trakTable file is consulted. Since the attributes of the ‘PolyData’ plot does not have information about the dataset that we are working on currently, this information needs to be passed to the ‘QvisPolyDataPlotWindow’ class. The information we are looking for is in the ‘CurrentFile.txt’ file (refer section 5.6 for more information about ‘CurrentFile.txt’ file) and we provide additional methods to read and parse the ‘CurrentFile.txt’ file in the ‘QvisPolyDataPlotWindow’ class methods.

5.3.2.1 New Methods in the PolyData Plot

The signature of the new methods added to the plot and their behavior is described below:

1) void CreateOpacityTable ( string, string );

   This method is called iteratively as many times as there are frames in the dataset. The input for this method is the object(s) to track and the location where the file has to be created. With the helper methods described below, the routine tracks the object(s) provided as an input and writes them to a file called ‘OpacityTable.txt’ (see appendix-I for file format).

The helper methods for creation of opacity table are as follows:

2) void ReadInfoForThisFile( );

   This method parses the ‘CurrentFile.txt’ file and extracts all the information in it.

3) bool startsWith ( string, string );
Checks whether a given string (a sentence) starts with the specified pattern and return true if it matches else returns false.

4) `string toString (int);`

This method converts a given integer type value into a string type value.

5) `vector<string> Tokenize (const string &);`

Given a constant reference to a sentence i.e. the string at the given memory location cannot be altered. This method breaks that sentence into words that are separated by a comma and returns those tokens as a vector of strings.

6) `void ReadCurrentFrameTrackInfo (const string &, const int &, map<string, string>&);`

This method reads the trakTable for a particular frame and creates a map of the objects in that frame. Every object in the frame is mapped to an object (or group of objects) in the next frame. Corresponding objects across the frames can be found easily by using this map data-structure [24].

7) `void SeparateCurrentAndNextFrameObjects (const vector<string> &, vector<string>&, vector<string> &);`

This method separates the entries in the TrakTable file into current frame and next frame objects.

8) `void PopulateNextFrameObjects (map<string,string>, vector<string>&, vector<string> &);`

This method creates a vector by referring to the map data structure and pushes all the objects for the next frame into a vector. It’s possible to have duplicate entries in the vector; we remove the duplicates from the vector to ensure that all the elements are unique.
5.5.3 Picking Objects with a Mouse Click

VisIt uses the Observer (also called subscriber – publisher) design pattern in which a Subject maintains a list of its dependents called the observers and notifies them automatically of any state change, usually by calling one of their methods [25]. VisIt provides support for picking objects in the ‘Zone Pick’ and ‘Node Pick’ modes (see Appendix – II for more details) and these modes subscribe to the ‘PickAttributes’. The ‘PickAttributes’ keep a track of current mouse coordinates and finds out whether the click is on area occupied by the object or is an empty space. Whenever there is a change in the ‘PickAttributes’, pick modes are notified of it via the UpdateWindow() method. This method is defined for every plot in the Qvis<Plotname>Window class. If the plot has to be made aware of changes in the ‘PickAttributes’, it needs to subscribe to the ‘PickAttributes’. Plots keep monitoring certain subjects; these are the attributes of the plot. Whenever there is a change in attributes, the plot is notified of it. To make our ‘PolyData’ plot aware of the ‘PickAttribute’ changes, we need to add one more subject i.e. ‘PickAttributes’ in the plot’s monitoring list. Whenever the ‘PickAttributes’ changes, the object number is extracted from the ‘PickAttributes’ by a method in the ‘PolyData’ plot’s UpdateWindow() method. This extracted object number is placed in the textbox for ‘Selective Feature Tracking’ (as described in previous section). In the previous section, the user entered this information by hand, now the process is automated. Once, the list of the features to be tracked selectively is formed, the user clicks on “Apply” and from there on it follows the same procedure as in previous section.

5.3.3.1 New Methods in the PolyData Plot

1) void UpdateObjectPicked (bool);
Whenever an object is picked, this method updates the ‘ObjNumToTrack’ widget (textbox) in the ‘PolyData’ plot attributes, with the current object number. It appends the current object number to already existing list of objects to be tracked.

2) `int PickedObject ()`;

The ‘PickAttributes’ has additional about the selected object. This method extracts the object numbers from this information and returns the object number to the calling method.

**Note:** VisIt starts object numbering from 0, whereas our numbering starts from 1, so to maintain consistency in our module, we add 1 to whatever object number VisIt selects. For example, if the object number of the object double clicked is 4, then we added 1 to it and display the object number as 5.

### 5.6 Technical Challenges with VisIt

**Unpredictable cycle time information**

One of the biggest hurdles towards development of custom plugins for time-varying datasets was the inability of VisIt to identify the current timestep. The working of VisIt can be broadly described as follows:

- Data is read from the files via a database reader.
- In case the user wants to do some processing, an operator is selected and the data passed on to plot for rendering on the screen.
- Now, the plot can make any additional changes before displaying the data on the screen.
Colormaps are generated for each frame and these are applied to the data by the ‘PolyData’ plot. If a colormaps are not applied to their corresponding frames, visualization results go awry. VisIt calls each frame as a cycle and employs a guessing mechanism to guess the cycle (or frame) number. If the guessed cycle number was used for choosing the corresponding colormap, there always was an error in the results rendered on the screen. Even though GUI displayed the cycle number correctly, plot was not getting the correct information resulting in inconsistent coloring while rendering the results on the screen. Hence, we devised a workaround that did not rely on VisIt’s guessing mechanism. This workaround is described in next section.

**Workaround**

VisIt automatically groups similar files in a folder as a database. If there are some .poly files in a folder on a computer, VisIt automatically group them together as a database. As can be seen in Figure 21, all the vorts poly files are grouped together as a vorts*.database. Double clicking any of these files opens the database and VisIt gets a corresponding cycle number by employing a guessing mechanism. All this functionality is provided by VisIt’s GUI module. One of the classes associated with VisIt’s GUI module is the QvisFilePanel. The QvisFilePanel class has a method called the UpdateFileSelection() that monitors the files in the database and reports whenever a new file is selected. A user can load any file in the list by double clicking on the file or using an animation slider to load the required file. This action would call the UpdateFileSelection() method. While making the database of the files, VisIt gather all the metadata information associated with the files in the database. From this metadata information, the location of the file (also called the Path), the number of files (frames) in the database, the starting & the ending
frame (or timestep) number, the current file (frame) and the name of the database are extracted. This information is written to a text file called the ‘CurrentFile.txt’ file (see appendix I for the format of this file). Now, VisIt waits for the user to add an operator or a plot and executes the network when the user clicks on ‘Draw’. The reader loads the data, and passes this information to the operator and the plot as the ‘vtkDataset’ object. The operator and the plot open the ‘CurrentFile.txt’ file, use the information in it, process the ‘vtkDataset’ object and pass on the information to the next stage. The ‘TrackPoly’ operator and the ‘Polydata’ plot get the location of the folder where the ‘CurrentFile.txt’ file is located from the attributes that the user enters. Whenever VisIt’s opens a new file, the ‘CurrentFile.txt’ file gets over written. VisIt loads the new data and issues an RPC to the operator and plot to work on this new data. As the RPC is issued only after the file is updated, the plot and the operator always have the updated information.
All the Files are grouped as a Database. All this information is part of QvisFilePanel methods.

Figure 22: Automatic database creation by VisIt. VisIt automatically groups the files of a dataset as a database. The methods to perform the automatic grouping are defined in QvisFilePanel class.
Chapter 6

Results

The Feature Tracking utility helps in studying the evolving patterns. We were mainly interested in tracking the following patterns:

1) Continuation of objects,
2) Merging to form new objects,
3) Bifurcation,
4) Dissipation,
5) Creation of new objects.

We ran the Feature Tracking implementation on the various datasets and tracked the above patterns in all of those datasets. The functionality of selective feature tracking and picking objects with a mouse click proved to be very useful. These functionalities helped to remove the visual clutter and focus on the features of interest. The results from application of the Feature Tracking on different datasets are given below:

1) **vorts dataset** – This simulation dataset of 100 frames (or timesteps) was generated on a Connection Machine (CM5) for studying problems of Computational Fluid Dynamics (CFD) [3]. The dataset contains vorticity and scalar vorticity magnitude that is computed for visualization and tracking. This simulation data is courtesy of N.Zabusky and V.Fernandez [27, 28]. Resolution of each frame is 128 * 128 * 128 and at each point a scalar value (float type) is stored. The results for application of feature tracking to visualize frames 5 to 10 of this vorticity dataset are given below in Figure 23. The visual analysis is possible because of a consistent coloring across frames, assisting the user in tracking objects. It’s very clear from the results that one
of the objects continues (encircled and labeled as continuation in figure 23) in all the frames. Also, two objects from frame 7 combine to form a bigger object; a new object is formed in this frame that continues to grow in the rest of the frames.

![Figure 23: Visualization Results for the Feature Tracking on vorticity dataset (vorts) from frames 5 to 10.](image)

We can clearly see the object encircled in frame 5 continuing in the rest of the frame. A new object is formed in frame 7 and keep on growing. Also, in Frame 7 two objects that were continuing as separate objects till now, combine to form a bigger object in frame 8, one of the objects disappear.

2) DNS Dataset – This dataset consisting of 27 frames is used in the study of hairpin packets of DNS data for Mach 3 turbulent boundary layer. Each frame of the dataset has a resolution of 384*256*69 and each point stores a scalar value (float type). We select a feature of interest in the first frame (object 45) and view its evolution for next few frames. The results for selective tracking of object 45 from DNS dataset are given in Figure 25. In order to get this dataset to work in VisIt, we had to write a
conversion program for converting the data to a format known to VisIt. We converted the files from .fld format (used by AVS) to .vtk (for VisIt) and the details of this program are given in Appendix – II.

Figure 24: Selective Feature Tracking of Object Number 45 in the DNS dataset.
3) **Surfactant molecular Dataset** (Modeling and Simulation Department, P&G) – This dataset of 101 frames is named volmap and the resolution of each frame is 47 * 47 * 47 points. A scalar value (float type) is stored at each of those points. We used Feature Tracking to study the feature evolution in the dataset. We produced the visualization results for first 8 frames of this dataset.

![Figure 25: Results from the application of the Feature Tracking & Visualization plugins on the volmap Dataset.](image)

4) **Indicator dataset (2D).** This dataset of 50 frames is a 2D dataset, each frame having a resolution of 384 * 256 points. Each point in this 384 * 256 frame represents a scalar value (float type). Our Feature Tracking software works only for 3D datasets, so we had to fake this dataset as a 3D dataset by adding an extra layer of Z component values. The format of the dataset could not be read by VisIt, so we had to convert it to .vtk format, and also add an additional Z layer to fake it is as a 3D
dataset. The program for this conversion is described in Appendix – II. We applied the Feature Tracking on this 50 frame dataset and show the results in Figure 22. In figure 22, we show the results from 4 frames (frame 1, frame 4, frame 10 and frame 20) out of 50 frames. Each frame now acts as a 3D volume and all actions like rotating the objects, selecting objects, and picking objects by mouse are valid. Next in Figure 27, we present the results of selectively tracking 2 objects from frame 1 to 50. Again, here we just show the results of 4 frames from frames 1 to 50.

Figure 26: Feature Tracking & Visualization results from four different frames of the Indicator (2D) Dataset.
Figure 27: Selective Feature Tracking of 2 objects from the Indicator (2D) dataset having 50 frames. We show results from four frames.

The results from the visualization of these datasets have been very similar to the ones produced from AVS. However, we could not make comprehensive performance calculations as both were running on machines having different hardware configurations. Newer versions of VisIt were not compatible on old Red Hat machines in the lab, so we had trouble compiling VisIt on Red Hat Linux 7.0. AVS could not be installed on newer machines because of license issues. Hence we could not run AVS and VisIt on the same system to compare the running times of the software packages. However, previous work has tabulated these comparisons and has shown that VisIt performs better than AVS [23].
Chapter 7

Conclusion

In this thesis we presented a methodology for Featuring Tracking & Visualization in VisIt. The objects in a frame were assigned colors based on the tracking information. Also, we provided an option to select the features of interest amongst all the extracted objects in a frame and track only those features of interest. We made our implementation more intuitive by providing the option of selecting the interesting objects with mouse clicks. We have documented the design philosophy of our plugins. This documentation can be useful for anyone creating new plugins in VisIt. Vizlab has developed a standalone distributed Feature Tracking algorithm. The current Feature Tracking & Visualization implementation in VisIt can be made more powerful by extending it to work in a distributed mode that will allow the handling of very large datasets. The next generation of Feature Tracking is aiming to provide a lot more functionality than before (like detection of super structures and so on) and this framework provides a solid foundation to build upon.
References

[22] VisIt supported File Formats, https://visualization.hpc.mil/wiki/VisIt_Data_Formats


# Appendix Table of Contents

## Appendix – I

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Installation of the Feature Tracking &amp; Visualization plugins in VisIt</td>
<td>89</td>
</tr>
<tr>
<td>Feature Tracking &amp; Visualization User manual</td>
<td>90</td>
</tr>
</tbody>
</table>

## Appendix – II

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>File Formats</td>
<td>100</td>
</tr>
<tr>
<td>Data structures in Scientific Visualization</td>
<td>110</td>
</tr>
<tr>
<td>Pick Modes</td>
<td>113</td>
</tr>
<tr>
<td>AVS to vtk converter</td>
<td>114</td>
</tr>
<tr>
<td>2D to 3D converter</td>
<td>117</td>
</tr>
</tbody>
</table>

## Appendix – III

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Feature Tracking related publications.</td>
<td>118</td>
</tr>
</tbody>
</table>
Appendix – I

Installation of the Feature Tracking & Visualization plugins in VisIt

VisIt has 3 main directories under the <visithomepath/src> folder. These directories are ‘operators’, ‘database’ and ‘plots’. Any new plugins should be created in one of these directories depending on the type of the plugin.

Automatic Installation

The source code for the Feature Tracking plugins can be downloaded from http://vizlab.rutgers.edu/FeatureTrackingCode/. Along with the source, an installation script is also available that will install all the plugins automatically. The steps for automatic installation using the installation script are as follows:

1. Download the Plugins.tar and InstallAllPlugins files to your <Yourvisithomefolderpath>.

2. Run the script from visit home folder, <Yourvisithomefolderpath> as follows:

   <Yourvisithomefolderpath>

   ./InstallAllPlugins

After successful installation, the new plugins should be visible in either the list of plots or the operators depending on the type of the plugin.

Manual Installation

The Feature Tracking plugins can also be installed manually. You need to create four folders for the plugins; two in ‘operators’ folder, one in ‘plots’ folder and one in ‘database’ folder. These folders should be named after the plugin name. The source code
for these plugins is available at http://vizlab.rutgers.edu/FeatureTrackingCode/. The steps
for compiling and installing the plugins individually is similar to the steps mentioned for
the installation of the ‘FeatureTrack’ operator in Chapter 5.

**Feature Tracking & Visualization User manual**

The Feature Tracking & Visualization plugins are divided into two groups. The first
group is for Feature Extraction & Tracking and the second group is for Visualization.

**Feature Extraction & Tracking**

Given a time-varying 3D dataset, the first step in the feature tracking process is to extract
the objects from the individual frames and track them. Application of the ‘FeatureTrack’
operator on the data results in generation of intermediate files that have information about
the extracted objects. Since an operator cannot be applied to the data without selecting a
plot, we select pseudocolor plot as the dummy plot. The ‘FeatureTrack’ operator
generates 5 files for each frame (or timestep). These file are .poly, .attr, .uocd. .trak,
colormap<frame#>.txt (refer to Appendix II for the format of each file). Also, a
.trakTable file is created that keeps track of the objects in the frame by recording the
interactions between the objects.

**FeatureTrack Attributes**

Figure 28 shows a screenshot of the attributes for the ‘FeatureTrack’ operator. Here, we
apply the ‘FeatureTrack’ operator for extracting and tracking the features from first 10
frames of the vorts dataset. The description of these attributes is as follows:

- **inputBaseFilePath** – is the location of the input dataset on the machine. The
  input dataset can be of any format that is supported by VisIt (.bov, .vtk etc). Along
  with the location of the dataset, the name of the dataset should also be
included for this attribute. For example, if the vorts dataset is at the location /home/data, then the entry for this ‘FeatureTrack’ operator attribute should be /home/data/vorts.

- **initialTimestep** – indicates the frame number (or timestep) from which the object segmentation should begin.

- **finalTimestep** – indicates the last frame in the feature extraction and tracking process.

- **percentThreshold** – is the threshold value used in calculations for extracting the features from the dataset.

- **smallestObjVolToTrack** – specifies the volume of the smallest objects that can be tracked by the code. If a user enters 1, it means that the volume of the smallest object that can be tracked is 1.

- **timestepPrecision** – indicates the number of ‘0’ s that should be added while combining the input base file path with the frame number to the form current frame name. If the frames of the dataset are stored as vorts0001.poly, vorts0002.poly, etc., it means the precision is 4. A precision of 1 indicates that no ‘0’ s are added to the frame number while forming the frame name.

![FeatureTrack operator attributes screenshot](image)

**Figure 28**: The FeatureTrack operator attributes screenshot.
Visualization

After the .poly files are created in the first step, the second step is to visualize those files. Figure 30 describes the order of the steps to visualize the .poly files. These steps are given below:

1. Select a .poly file by either double clicking on the file or selecting the entire auto-generated .poly database.
2. Click on the ‘Plots’ tab in GUI (Figure 30). This shows the list of available plots for the dataset. Select ‘PolyData’ plot from the list.
3. Click on the ‘operators’ tab in GUI (Figure 30). This shows the list of available operators for the dataset. Select ‘TrackPoly’ operator from the list.
4. Select the attributes for the selected plot (Figure 29) by clicking on ‘PlotAtts’ tab and click on ‘Apply’. Similarly select attributes for operators (Figure 29) by clicking on ‘OptAtts’ tab and click on ‘Apply’.
5. Click on “Draw” to get the results, as shows in Figure 30.

TrackPoly Attributes

- **PolyFilePath** – this attribute specifies the path (or the location) of the .poly files on the machine.

- **InitialColorScheme** – lets the user choose between random or user-defined color scheme. If the ‘Random’ option is selected, then the colors are assigned randomly to the objects in the first frame. In subsequent frames, the objects are assigned colors based on these random colors and tracking information from the trakTable file. When the user selects the ‘UserSelectedFile’ option, the objects in the frame are assigned colors according to the criteria mentioned in the file specified by the user. Colors to the objects in the subsequent frames are assigned based on the
color information in the user specified file and the tracking information from the trakTable file.

- **SelectedFile** – this attribute specifies the location of the user-defined colormap file on the machine.

![Figure 29: The PolyData and TrackPoly Attributes screenshot.](image)

**PolyData Attributes**

- **Path** – specifies the location of the .poly files on the machine.

- **SelectiveObjTracking** – with this option, the user can choose to visualize only a specific group of object(s) and ignore the rest of the objects in the frame.

- **ObjectNumbertoTrack** – is a textbox that is activated when the user selects selective object tracking option. In this textbox, the users can enter the object
numbers separated by a comma. If nothing were entered in the textbox, all objects will be tracked.

- **OpacityForOtherObjects** – with this opacity slider, the user can select the opacity for the background objects. The default value is 5% and it can be changed to any value between 0 and 100% via the slider.

Figure 30: Visualizing the .poly file created by the ‘Feature Extraction & Tracking’ module. (1) Double click on the .poly file as indicated in the figure. (2) Click on ‘plots’ and select ‘PolyData’ plot from it. (3) Click on ‘Operators’ and select ‘TrackPoly’ from the list of operators. (4) Fill in the ‘plot’ and ‘operator’ attributes. (5) Click on ‘Draw’ to render the vorts1.poly file.
Selective Feature Tracking

Figure 31 shows the ‘PolyData’ (plot) and the ‘TrackPoly’ (operator) attribute windows. When a user wants to track only some of the objects, the rest of the objects in the frame have to be made transparent. The steps involved for this process are:

1) Click on the ‘Yes’ checkbox for the label ‘SelectiveObjTracking’.

2) Enter the object numbers for the objects that are to be tracked. This can be entered either manually (if you know the object number) or you can double click on an object and its object number would be extracted automatically. Note: This happens because of the functionality we added while designing the Feature Tracking & Visualization plugins and VisIt should be either in ‘Zone pick’ or ‘Node pick’ mode.

3) Select the opacity for the objects being tracked. ‘Opacity’ slider controls the opacity of the objects that are being tracked. The default value for the opacity of the selected objects is 100%.

4) Select the opacity for other objects in the frame. ‘Opacity For Other Objects’ slider controls the opacity for the other objects in the frame. This slider has a default value of 5%.

After following the steps mentioned above, click on ‘Apply’ in the plot attribute window. Now, only the objects in the list are rendered as opaque objects, rest of the objects become transparent. Figure 32 displays the results for selectively tracking a single object (object No.5). We chose opacity value as 5% for rest of the objects in the frame.
Figure 31: The ‘TrackPoly’ operator and the ‘PolyData’ plot attribute window. The object numbers of the objects to be tracked are entered in the textbox (circled in red). Here, we track only one object i.e., object number 5. The ‘OpacityForOtherObjects’ slider (circled in blue) can control the opacity for the rest of the objects in the frame that are not being tracked.
Picking Objects by Mouse clicks

Selective feature tracking is a useful functionality that lets the users track only some of the objects mentioned in a list and provided as an input to the program. The list containing the object numbers is built manually. For a frame with many objects, it becomes difficult to remember the object numbers for all the objects in that frame. ‘Label’ plots can be used to find out the object numbers for the objects in a frame. But, this process is tedious and error prone. The most intuitive behavior in such situations is to be able to select the objects with mouse clicks. In this thesis, we add this functionality to the Feature Tracking module. For selecting objects with mouse clicks, first a pick mode is chosen. This can be either the ‘Zone Pick’ or ‘Node Pick’ mode and whenever an object
is selected, a new ‘Pick’ window (Figure 30) pops up (see appendix II for more info on pick modes). Another change that can be seen is in the ‘PolyData’ plot attribute window. The textbox associated with the ‘ObjectNumToTrack’ label will display the object number of the selected object. If the mouse click does not correspond to any object, a message is displayed on the screen and nothing gets added to the list of objects being tracked.

Figure 33: Picking objects with mouse clicks. The first step is the selection of a pick mode (Zone Pick or Node Pick), we select ‘Zone Pick’ mode in this case. Then we select two objects, one is on the left and the other one is on the right hand side of the image. Those objects are labeled as Object A and Object B respectively.
Instead of manually entering the object numbers, now we generated this list by clicking on interesting objects. To see the results, click on ‘Apply’ in the ‘PolyData’ plot attribute window. From here, the process is same as the ‘Selective Feature Tracking’ process and you get results as shown in figure 32.

Figure 34: Pick Information window: on selecting an object with a mouse click, a new ‘Pick’ window pops up. This window has information about the picked objects. The object is automatically labeled, as A. Changing the attributes in ‘Pick’ window can change this style of labeling. Also, the object number of the picked object is added to the textbox ‘ObjectNumberToTrack’ (as circled) in the PolyData plot attribute window.
Appendix – II

File Formats

.poly file

This file is written for each frame (or timestep) in the dataset. It contains node coordinates of the polygons, vertices and their connectivity information for the iso-surface visualization. The format of the .poly file is given below:

< red > < green > < blue >
< numnodes >
< x0 > < y0 > < z0 >
< x1 > < y1 > < z1 >
< x2 > < y2 > < z2 >
< x3 > < y3 > < z3 >
< x4 > < y4 > < z4 >
.
.
< numof connections >
3 < vertex ID > < vertex ID > < vertex ID >
3 < vertex ID > < vertex ID > < vertex ID >
3 < vertex ID > < vertex ID > < vertex ID >
.
.
0   // signifies end of information about an object in the frame
< red > < green > < blue >

// continues for next objects, followed by 0 and so on.

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
An .attr file is generated for each frame of the dataset, and contains information (centroid, moments, max node value etc) about each object in the frame. An .attr file looks like this:

Object <obj ID> attributes:

Max position: (<max X>, <max Y>, <max Z>) with value: <max node value>
Node No.: <num node min>
Min position: (<min X>, <min Y>, <min Z>) with value: <min node value>
Node No.: <num node max>
Integrated content: <integrated content>
Sum of squared content values: <sum squared content value>
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

This file is generated for each frame of the dataset. It contains information about the frame, particularly the number of objects, cell information etc. This file is useful for volume rendering. The format of the file is given below:

<time>
<num objects>
<obj ID>       // starts from 0
<volume> <mass> <centroid X > < centroid Y> < centroid Z> // volume is of number of points
<lxx> <lyy> <lzz> <lxy> <lyz> <lzx>
<point ID> <point X> <point Y> <point Z> <point value> // point ID starts from 0
<point ID> <point X> <point Y> <point Z> <point value>
<point ID> <point X> <point Y> <point Z> <point value>
.
<obj I D>
.

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
49
1798593 65.000000 99.000000 109.000000 10.520865
.
1
.

.trak file

This file is also generated for each frame of the dataset. This is like a high level summary of the objects in the frame. It contains the number of objects and the number of nodes to be used for memory allocation for tracking with successive timestep. The format of this file is given below:

<file base name> <time> <mass> <volume> <centroid X> <centroid Y> <centroid Z>
<file base name> <time> <mass> <volume> <centroid X> <centroid Y> <centroid Z>

Sample .trak file

/home/admin/data/GENERATEDTRACKFILES/vorts10 10.000000 83432.195312
1263768.521065 97.046165 99.475235
/home/admin/data/GENERATEDTRACKFILES/vorts10 10.000000 42605.246094
631662.313812 117.506172 39.228920
/home/admin/data/GENERATEDTRACKFILES/vorts10 10.000000 40073.558594
577162.039474 46.921017 4.576792

.trakTable file
There is only one .trakTable file for the entire dataset. This file is not created if there is
only one frame in the dataset. Otherwise this file is created from the second frame and
with each new frame, the tracking information gets appended to the end of the file. The
trakTable file gives the evolution history of each object frame by frame along with an
indication of any events that occur. The format of the file is like this:

<Frame #>
<Objects of Currents Frame> -1 <Corresponding objects in next frame>

‘-1’ acts as delimiter to help identify between objects of current frame and next frame. If
the objects of the current frame are disappearing in next frame, there is nothing to the
right of -1. If new objects are forming in next frame, then there would nothing to left of -
Sample .trakTable file

Frame #3

1 5 7 -1 2 5 // 3 object from frame 2 (1, 5, 7) combine as objects 2 & 5 in frame 3
6 -1 1 10 // Object 6 from frame 2 splits into Object 1 & 10 in Frame 3
2 -1 3 // object 2 continues as object 3 in next frame
4 -1 4
-1 6
-1 7 // this is a new object created in Frame 3
-1 8
-1 9
3 -1 // this object no longer continues in Frame 3, it dissipates.
8 -1

colormap.txt

This file is generated for each frame of the dataset. The .poly file contains information about the color of the object. VisIt allows to assign different colors to the objects than the ones specified in the .poly files. The information about the colors of the object is passed
to VisIt and VisIt constructs a color map from this information. The colormap.txt file contains red (R), green (G) and blue (B) intensities values and an alpha value (opacity) for each object in the frame. R, G and B are integer values in the range 0 to 255 and alpha values are floating values between 0.0 and 1.0. The number of entries in this file corresponds to the number of objects in the frame. It looks like this:

`colormap< Frame number>.txt`

```
<R_i> <G_i> <B_i> <alpha_i>      // object 1
<R_2> <G_2> <B_2> <alpha_2>      // object 2
.
.
.
<R_n> <G_n> <B_n> <alpha_n>      // object n
```

**Sample colormap file** for frame 5 with 5 objects. It’s named as colormap5.txt.

```
10  200  50  1.0
78  110  240  0.8
0  0  255  1.0
255  255  50  1.0
75  75  75  0.5
```

**CurrentFile.txt**

This file is updated every time a new file is loaded by VisIt with the help of a suitable database reader. This file records the information about the current file that is being used by VisIt. VisIt automatically creates a database for the related files of a dataset and this
file has all the information about that dataset, for example, the starting and the ending frame (or file) numbers, the cycle number or the current frame (or file opened) and the number of steps to reach from the start cycle to the end cycle. This file also stores the name of the database (from the file name).

**Sample CurrentFile.txt**

```plaintext
CURRENT_FILE=/home/admin/data/vorts1.bov
PATH=/home/admin/data/
START_CYCLE=1
END_CYCLE=10
CYCLE=1
STEP=10
DATASET_NAME=vorts
```

curpoly.txt

This file is generated when the ‘Poly’ database reader reads a poly file. Once this file is created, it’s modified by the TrackPoly operator to indicate the current frame number. Other than the frame number, this file also records the name and the number of objects in the frame. This file is just to double check whether the current frame number is conveyed properly or not, as the cycle number in “CurrentFile.txt” also represents this information. This way, we can be sure that the ‘PolyData’ plot gets the correct colormap information.

**Sample curpoly.txt**

vorts1.poly // File name
OpacityTable.txt

This file is generated only if the selective tracking option is enabled. There is only one file for the entire dataset. This file has information about the non-opaque objects in a frame and for every frame the object number of the non-opaque objects gets appended to this file. This is basically the tracking information for the objects mentioned in starting frame.

Sample OpacityTable.txt file

NonOpaqueObject_Frame #1
4
6
NonOpaqueObject_Frame #2
3
5
8
NonOpaqueObject_Frame #1
3
4
5
8
Data structures in Scientific Visualization

A big hurdle in using any visualization system is getting the data into the software package. The form of the geometry in the problem frequently dictates the form of the data: e.g., a spherically symmetric problem results in spherical-polar coordinates in the data set. Each visualization package has defined a set of data types upon which the algorithms depend to function. Now we will see the basic data structures and the way visualization packages sees them.

Uniform Mesh

A uniform mesh has grid spacings that don't change along the axes. To specify the grid along one axis any three of the four values (min. value, max. value, step size and number of steps) must be specified, and we can find the last value. With three of the above values for each axis the software package can construct the mesh and generate connectivity information.

Figure 35: An Uniform Mesh (Image source – http://sciviz.aict.ualberta.ca/index.php?page=data&sub_page=structure)
**Rectilinear Mesh**

A rectilinear mesh has orthogonal axes and the grid coordinates are specified along each axis, for 2D datasets, we must provide X, Y coordinates and for 3D datasets Z coordinates should also be included.

![Rectilinear Mesh](http://sciviz.aict.ualberta.ca/index.php?page=data&sub_page=structure)

**Irregular Mesh**

An irregular mesh has less symmetry than uniform or rectilinear meshes. The coordinates of all points in the mesh must be provided to the visualization software, however, the connectivity between the nodes can be inferred.

![Irregular Mesh](http://sciviz.aict.ualberta.ca/index.php?page=data&sub_page=structure)
Structured Mesh

In the terminology of AVS/Express uniform, rectilinear, and irregular fields all exist as structured meshes. They consist of nodes arrayed on a one-, two-, or three-dimensional hexagonal grid. This is all the information that is required to know how the nodes are organized and connected in space.

![Structured Mesh](http://sciviz.aict.ualberta.ca/index.php?page=data&sub_page=structure)

Unstructured Mesh

The unstructured mesh is the only type that must have the connectivity information provided to the module as an input. In unstructured meshes (where cell data is unstructured), nodes exist in space, but as part of a higher organization called a cell. The nodes are connected together in a certain order to form point, line, triangle, quadrilateral, tetrahedral, pyramid, prism, or hexahedral cells. These cells collectively form the field.

![Unstructured Mesh](http://sciviz.aict.ualberta.ca/index.php?page=data&sub_page=structure)

The language used to describe this mesh varies between the packages as follows:
<table>
<thead>
<tr>
<th>Mesh Type</th>
<th>AVS</th>
<th>Vtk (or VisIt)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Uniform</td>
<td>Uniform Mesh</td>
<td>Structured Point</td>
</tr>
<tr>
<td>Rectilinear</td>
<td>Rectilinear Mesh</td>
<td>Rectilinear Grid</td>
</tr>
<tr>
<td>Irregular</td>
<td>Irregular Mesh</td>
<td>Structured Grid</td>
</tr>
<tr>
<td>Structured</td>
<td>Structured Mesh</td>
<td>Structured Grid</td>
</tr>
<tr>
<td>Unstructured</td>
<td>Unstructured Mesh</td>
<td>Unstructured Grid</td>
</tr>
</tbody>
</table>

**Pick Modes**

VisIt provides a way to interactively pick values from the visualized data using the visualization window’s Zone Pick and Node Pick modes. When the visualization window is in one of these pick modes, each mouse click in the visualization window causes VisIt to find the location and values of selected variables at the pick point. When VisIt is in Zone pick mode, it finds the variable values for the zones that you click on. When VisIt is in node pick mode, similar information is returned but instead of returning information about the zone that you clicked on, VisIt returns information about the node closest to the point that you clicked. Pick is an essential tool for performing data analysis because it can extract exact information from the database about a point in the visualization. You can put the visualization window into one of VisIt’s pick modes by selecting Zone Pick or Node Pick from the Popup menu’s Mode submenu. After the visualization window is in the pick mode, each mouse click causes VisIt to determine the values of selected variables for the zone that contains the picked point or the node closest to the picked point. Each picked point is marked with an alphabetic label, which starts at A, cycles through the alphabets and repeats. The pick marker is added to the visualization window to indicate where pick points have been added in the past. To clear pick points from the
visualization window, select the Pick point option from the Clear menu in the Main Window’s Window menu. The dimension of the plots in the visualization does not matter when using pick mode. Both 2D and 3D plots can be picked for values. However, when using pick mode with 3D plots, only the surface of the plots can be picked for values. If you want to obtain interior values then you should use one of the Pick queries or apply operators that expose the interiors of 3D plots before using pick.

The Pick Window mainly consists of a group of tabs, each of which displays the values from a pick point. The tab labels A, B, C, etc. corresponds to the pick point label in the visualization window. Since there are a fixed number of tabs in the Pick Window, tabs are recycled as the numbers of pick points increases. When a pick point is added, the next available tab, which is usually the tab to the right of the last unused tab, is populated with the pick information. If the rightmost tab already contains pick information, the leftmost tab is recycled and the process repeats. The information displayed in each tab consists of the database name and timestep, the coordinates of the pick point, the zone/cell that contains the pick point, the nodes that make up the cell containing the pick point, and the picked variables. The rest of the Pick Window is devoted to setting options that format the pick output.

**AVS to vtk converter**

VisIt cannot load the data from the .fld files. Hence, either a database reader module or a conversion program had to be written. In this thesis, we wrote a program to convert the .fld files into the .vtk format. The source code for this conversion program can be downloaded from http://vizlab.rutgers.edu/FeatureTrackingCode/.
Sample .fld file

# AVS field file

ndim=2            # number of computational dimensions in the field
dim1=7            # Dimension of axis 1
dim2=4            # Dimension of axis 2
#dim3=1            # Dimension of axis 3
nspace=2          # number of physical coordinates at each point
veclen=1          # number of vector components at each point
data=float        # the data in the file is of type: byte, integer, float or double
field=uniform     # uniform, rectilinear, irregular
min_ext= 0  0     # lower bounds of x, y and z axes
max_ext= 6  3     # upper bounds of x, y and z axes
label=radius      # label the field components
unit=arbitrary    # units for the field components

# For a uniform field read in coords of points with min and max values
coord 1 file=unif_2D.dat filetype=ascii skip=11 offset=0 stride=3
coord 2 file=unif_2D.dat filetype=ascii skip=11 offset=1 stride=3
#coord 3 file=unif_2D.dat filetype=ascii skip=11 offset=2 stride=3

# Read variables in like this - read data values in 4th column
# Don't need the coordinate data in the first three columns
variable 1 file=unif_2D.dat filetype=ascii skip=13 offset=3 stride=4

Sample .vtk file

vtk DataFile Version 1.0
Structured Points 3D Dataset

ASCII

DATASET STRUCTURED_POINTS

DIMENSIONS 7 4 1

ORIGIN 0 0 0

SPACING 1 1 1

POINT_DATA 28

SCALARS scalars int

LOOKUP_TABLE default

1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 1

The conversion program parses the .fld file and creates a corresponding .vtk file. Both the files contain similar information; the only difference is the way coordinates are specified. In .fld files, the coordinates can be mentioned in a separate binary file, whereas this is not possible in .vtk files. All the coordinates must be specified in .vtk file and not in a separate file. The data values can be either in ASCII or Binary mode.
2D to 3D converter

Our Feature Tracking implementation does not work on 2D datasets. For tracking features in 2D datasets with our implementation, 2D datasets had to be faked as a 3D dataset by adding an additional layer of points on the Z axis, equal to the product X & Y axis coordinate points. Each point in the additional layer is given a data value that is the minimum value in the dataset. The converter code consists of a section to parse the .fld file and convert it to the .vtk file format (similar to the one discussed in previous section) and another section to add an extra Z layer to the dataset. In .vtk files, 2D datasets have Z coordinate value as 1. To fake the dataset as 3D, the value of Z coordinates was changed to 2 and additional points were added to under the section LOOKUP_TABLE (the format of .vtk file is given in previous section) of .vtk file. Now, we can track features in this time-varying 2D dataset that is faked as a 3D dataset. The source code can be downloaded from http://vizlab.rutgers.edu/FeatureTrackingCode/.
Appendix – III

**Feature Tracking related publications.**

Here, we list all the Feature Tracking related publications from Vizlab students. Also, given in this appendix are the list of PhD and Master’s thesis related to the development of the Feature Tracking algorithms and the implementations in various visualization software packages. The publications related to the Feature Tracking are listed below:


List of the thesis related to the Feature Tracking are listed below:


