Seventh International Conference on Computational Fluid Dynamics (ICCFD7), Big Island, Hawaii, July 9-13, 2012

ICCFD7-1703

IFDT – Intelligent In-Situ Feature Detection, Extraction, Tracking and Visualization for Turbulent Flow Simulations

Earl P.N. Duque[1] , Daniel Hiepler[2], Christopher P. Stone.[3] and Steve M Legensky[4]
Intelligent Light, Rutherford, New Jersey, 07070
and
Kwan-Liu Ma[5], Christopher Muelder [6] and Jishang Wei[7]
College of Engineering, Department of Computer Science,
University of California, Davis 95616

Abstract: Intelligent In-Situ Feature Detection, Tracking and Visualization for Turbulent Flow Simulations (IFDT) is a new prototype visualization and CFD data analysis software system for flow feature data tracking and extraction. The system utilizes volume rendering with an Intelligent Adaptive Transfer Function that allows the user to train the visualization system to highlight flow features such as turbulent vortices. A feature extractor based upon a Prediction-Correction method then tracks and extracts the flow features and determines the statistics of features over time. The method executes In-Situ with a flow solver via a Python Interface Framework to avoid the overhead of saving data to file. This prototype system enables the user to readily explore, detect, track and analyze flow features predicted by large scale unsteady CFD simulations.

Keywords: Feature Detection, Computational Fluid Dynamics, Turbulence, Visualization, In-Situ, Feature Tracking.

1. Introduction

Large Scale scientific and engineering physics based simulations have become more attainable with the advent of more affordable computing. Whereas ten years ago large scale computations were steady and on the order of 10's of millions of grid points, today's large scale computations are unsteady and on the order of 100's of millions and exceeding billions of grid points. Furthermore, large scale computations may contain a wealth of information as these computations can now better resolve the turbulent behavior of the fluid, simulate multiple species due to mixing and reactions and capture complicated shock interactions.

Better tools that can automatically detect pertinent flow features are needed. Contemporary open source and commercial CFD Post-Processing software have built-in functionalities to automatically detect such flow features. For instance, FieldView [1] has the capability to compare different data sets using its Data Set comparison tool. In addition, it has the capability to automatically detect vortex

[1] Manager, Applied Research Group, 301 Rt 17N, 7th Floor.
[2] Research Engineer, Applied Research Group, 301 Rt 17N, 7th Floor.
[3] Research Engineer, Applied Research Group, 301 Rt 17N, 7th Floor.
[4] General Manager & Founder, Intelligent Light, 301 Rt 17N, 7th Floor.
[5] Professor of Computer Science, University of California-Davis, 1 Shields Avenue, Davis, California
[6] Post-Doctorate Researcher, University of California-Davis, 1 Shields Avenue, Davis, California
[7] Graduate Research Assistant, University of California-Davis, 1 Shields Avenue, Davis, California

cores and separation and reattachment lines using either vorticity or eigenvalue analyses by Haimes [2]. However, these tools and other feature detection tools as highlighted by Thompson [3] were developed for steady state flow fields. These methods lack the ability to sample and explore extensive unsteady data sets. They furthermore rely upon visualization techniques that compare images and require excessive user intervention. New tools are needed that allow for more straightforward feature extraction to yield better physical understanding of turbulent and inherently unsteady flow phenomena which will help one to predict and understand key physics in all speed regimes.

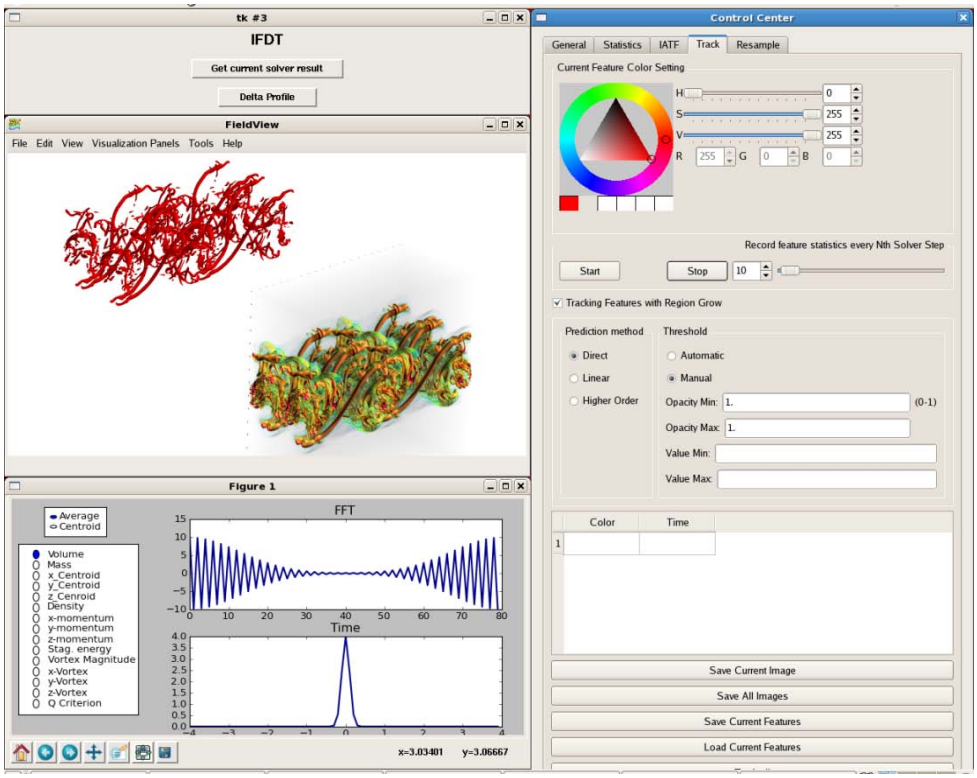


Figure 1: IFDT User Interface

The new prototype visualization system "Intelligent In-Situ Feature Detection, Tracking and Visualization", (IFDT) presented herein addresses these needs. This method utilizes a Python-based Software Interface Framework that enables a Python wrapped flow solver to share, without I/O penalties, the pertinent data structures and allows it to be controlled from the familiar (but modified) FieldView user interface shown in Figure 1. The user interface provides an interactive graphical front end tool that allows knowledgeable domain experts (i.e. combustion turbulence or structural aero-acoustics experts) to identify and pick flow features rendered in a few initial time steps of the solution. Once identified the system automatically tracks those flow features in time and gathers pertinent statistics of that feature.

2. Software Components

IFDT consists of the flow solver (currently LESLIE3D), a Python Software Interface Framework, Predictor-Corrector Feature Extraction and Tracking (PCFET), and Intelligent Adaptive Transfer Function (IATF). In addition IFDT includes a modified version of the FieldView version 13 client that supports Direct Volume Rendering, has prototype Python based graphical user interfaces (GUI) to control PCFET and IATF and In-Situ FieldView servers that can share memory space with the CFD solver; avoiding file I/O. The following sub-sections describe in detail each of these components.

2.1. Flow Solver LESLIE3D

The flow solver "Large- Eddy Simulations with Linear-Eddy Model in 3D (LESLIE3D) by Menon, et.al [4] is a research level Computational Fluid Dynamics (CFD) code used to investigate a wide array of turbulence phenomena such as mixing, combustion, and acoustics. It is a structured-grid CFD solver that incorporates Large-Eddy Simulations (LES) methods with various high-fidelity mixing and combustion models. LESLIE3D has been used to model a very wide range of turbulent flow problems including gas turbine combustors [5] and super-critical liquid fueled rocket engines [6]. LESLIE3D has also been used to model more fundamental flow physics such as premixed flame dynamics in isotropic turbulence [7].

LESLIE3D solves the compressible form of the governing Navier-Stokes or filtered LES equations using a multi-block structured-grid finite-volume scheme. The 2^{nd}-order accurate explicit McCormack predictor-corrector scheme is used to solve unsteady problems. LESLIE3D is capable of either 2^{nd} or 4^{th}-order spatial accuracy for both the viscous and inviscid terms. The turbulence kinetic energy equation (k-equation) is solved when turbulence modeling is required. Block-decomposition with MPI is employed for distributed parallel computing.

A reduced version of the larger LESLIE3D code was used in the IFDT in-situ coupling process. This version of the code, 107.leslie3d, is distributed as part of the SPEC CPU2007 Benchmark suite of codes designed to measure scalar and parallel hardware performance on CPU-intensive applications [8]. This benchmark code is specially designed to solve the unsteady evolution of a mixing shear layer that can be modeled with a single Cartesian computational grid in either 2- or 3-dimensions and with a wide range of grid resolutions. Further, it can be initialized and validated analytically without requiring any file I/O making it an ideal candidate for scaling and performance studies.

2.2. Python Software Interface Framework and In-Situ Processing

LESLIE3D and the feature tracking components were coupled with FieldView based upon the Python Software Interface Framework (SIF) described by Sitaraman, et.al. [9]. The SIF is a framework in which multiple tools such as a solver, post-processor and feature detector share the same address space passing pointers to memory instead of writing and reading datafiles; avoiding bottlenecks from having to read and write large data files. To operate within the SIF, each program is converted to a shared library and execute under a Python high level control.

Figure 2 presents a schematic on how the in-situ feature tracking works in IFDT. Here the flow solver (FS) and the "Predictor Corrector" based Feature Tracking tool (PC), data resampler (RS) and the FieldView server processes (fvsrv) execute all on the same server nodes (Node 1….N). The server machine can be a distributed processor system accessed across a wide area network as shown or a shared memory machine with all processes and communication occurring local to one compute node. All the components are Python wrapped which allows them to share data structures via the Python based execution interface. For polygonal surfaces, fvsrv points and copies the nodal based data, creates surface extracts and then sends it to the client for rendering. For Direct Volume Rendering, volume data extracts are sent from the RS to the fvsrv to the master fvsrv at Node 0 for interactive viewing and analysis by the user on the client. In either case, the user has full functionality of FieldView and its standard polygon based graphics such as iso-surfaces, boundary surfaces, 2-D plots.

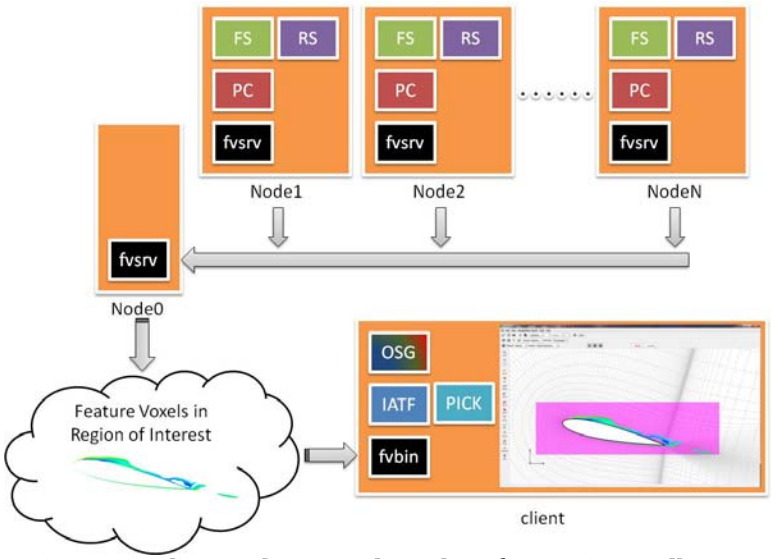


Figure 2 – Client and Server Flow Chart for In-Situ Intelligent Feature Detection and Tracking

For volume rendering, voxel information is collected from all the server nodes and sent to the client computer. Each In-situ FieldView server sees only its portion of a single large grid being solved by LESLIE3D. When the FieldView client requests voxels it specifies a bounding box (Region of Interest – ROI) within the single large grid. The bounding box is sent to all the FieldView servers which then send a portion of the original grid they hold and return the intersection bounding box.

The In-situ capability of IFDT relies upon the ability to directly and efficiently access CFD solution data without resorting to file I/O. In the current in-situ IFDT design, the parallel CFD solvers and FieldView servers run concurrently on separate threads within the same Linux process. This design allows the FieldView server to directly access solution data since they share the same memory space.

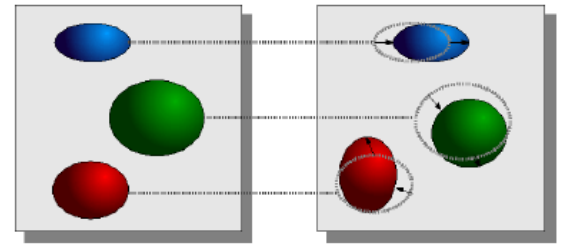
The FieldView server allocates memory for the xyz grid, iblank, solution q and function-array. For simplicity, let us assume an equal grid size (N) in the IJK grid directions; therefore, each FieldView server would allocate 3 (xyz) + 1 (iblank) + 5 (q) + 5 (function) = 14 arrays with size N^3. The LESLIE3D finite-volume scheme requires approximately 110 double-precision arrays with additional 3-layers of ghost-cell data. For large values of N, the ghost-cell data contribution can be ignored. In this situation, **the FieldView server memory only adds 7% to the total memory footprint**. LESLIE3D's storage requirement is higher than most production-level CFD codes so the relative overhead from the FieldView server is likely higher for other codes. For example, NASA's **OVERFLOW-2** [10] solver requires approximately 49 N^3 arrays, ignoring the added storage required by the overset grid algorithm. This equates to approximately a **14% overhead for the FieldView server storage**.

The in-situ IFDT direct Volume Rendering based visualization processes currently require uniform Cartesian data. Since most CFD simulations of interest use curvilinear or unstructured grids, the underlying simulation data within the user-defined *region-of-interest* (ROI) is interpolated onto a uniform Cartesian mesh. Therefore, the FieldView server threads also perform an interpolation process. This process, known as *re-sampling*, adds some measure of memory and computational overhead to the CFD solver. By virtue of the full-copy extraction method, the re-sampling operation can proceed concurrently to the CFD solver reducing the negative performance impact. The worst-case scenario would occur when the re-sampling and the CFD resolution matched. In this situation, the re-sampling algorithm increased the FieldView server overhead to 9% with LESLIE3D. However, one of the design features of the re-sampling process is to allow a coarser visualization mesh. Using half the CFD resolution reduces the total number of re-sampling points by 1/8, nearly an order of magnitude, and thus, imposes a negligible memory overhead.

The Volume Rendering occurs on the FieldView Client whereby all the re-sampled data is assembled on the master FieldView server then sent to the Client. This design imposes a restriction upon the size of the re-sampling mesh; it must be small enough to fit on the single FieldView master server computer. However, the re-sampling and feature tracking algorithms only consider a single scalar value. Further, the re-sampled mesh is Cartesian and requires only 36 bytes to be fully defined, a considerable savings compared to explicitly storing the xyz coordinates of the re-sampled mesh. The storage for one array is quite small relative to that required by most CFD solvers. For example, **a 512^3 simulation, over 100 million grid points, only requires 512 MB on the FieldView master**; a small memory overhead on modern systems. As stated earlier, the re-sampling algorithm allows the user to control the post-processing cost by adjusting the ROI location and resolution.

2.3. Predictor-Corrector Feature Extraction and Tracking

The Feature Extraction and Tracking method deployed in IFDT is based upon the prediction-correction method by Muelder and Ma [11]. This method uses a prediction step to make the best guess of the feature region in the subsequent time step, followed by growing and shrinking the border of the predicted region to coherently extract the actual feature of interest. They utilize information from previous time steps to predict a feature's region in the current time step, then correcting the predicted region to extract the feature's actual region through growing and shrinking processes. As shown in Figure 3 the approach works by predicting a feature's region in subsequent time steps then extracting the actual region starting with this predicted region. By doing this it effectively reuses the information from previous time steps to reduce the amount of work needed in each time step. This method allows for the extraction of individual features without the need to do a global segmentation where all features are extracted at once. This capability gives it the interactive response needed for the IFDT system. The user can dynamically pick or select features for extraction and tracking and dynamically adjust parameters and alter the volume being tracked.



(b) Our approach: Predict feature's region based on previous time steps then reshape it to the correct region

Figure 3: Feature tracking approach utilizes coherency between time steps in the extraction (Taken From: Muelder, C. and Ma, K.L, "Interactive Feature Extraction and Tracking By Utilizing Region Coherancy", [11]

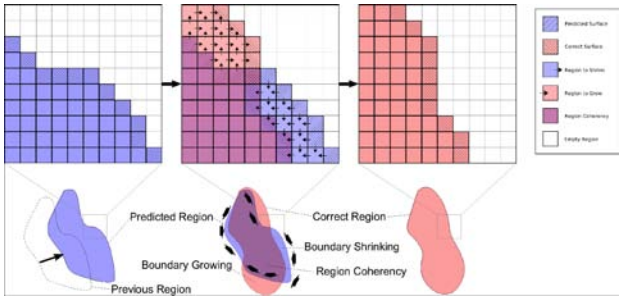


Figure 4 - Coherent extraction. First, the feature in the previous time steps is used to predict the feature's region in the current timestep. However, the predicted region of the feature is not the correct region. The region is then corrected by shrinking and growing the boundary (Taken From: Muelder, C. and Ma, K.L, "Interactive Feature Extraction and Tracking By Utilizing Region Coherancy", [11]

The technique combines the prediction of a feature's region and then corrects by adjusting the surface of this region. Prediction is the process of guessing a feature's region in space based on its location in previous time steps. Once this prediction is made, then the actual region is extracted by adjusting the surface of the predicted region through the use of a technique based on region growing. When predicting a region for a time step, the prediction will almost never line up exactly with the correct region. Thus, it is necessary to correct the region by adjusting its boundary as

necessary. As depicted in Figure 4, this adjustment is actually a combination of two processes: boundary growing and boundary shrinking. Both boundary growing and shrinking are based on region growing [12].

2.4. Intelligent Adaptive Transfer Function IATF

When extracting and visualizing time dependent flow features in large volume datasets, the values of the scalar functions defining the flow features (i.e. vorticity magnitude, Q-Criterion or Lambda-2) may change dramatically as the flow evolves in breadth (min-max values) and absolute magnitude. These variations make it difficult for a user to determine the correct scalars to choose that will highlight the flow features and enable a successful feature extraction.

To overcome these issues, the Intelligent Adaptive Transfer Function method by Tzeng and Ma [13] was adopted. Whereas conventional methods require either an analytical description of the feature of interest or tedious manual intervention throughout the feature extraction and tracking process, they showed that it was possible for a visualization system to "learn" to extract and track features in a complex transient flow field according to their "visual" properties, location, shape, and size. The basic approach was to employ machine learning in the process of visualization. The power of such an intelligent system approach is its ability to extract and track a feature of interest in a high-dimensional space without explicitly specifying the relations between those dimensions, resulting in a

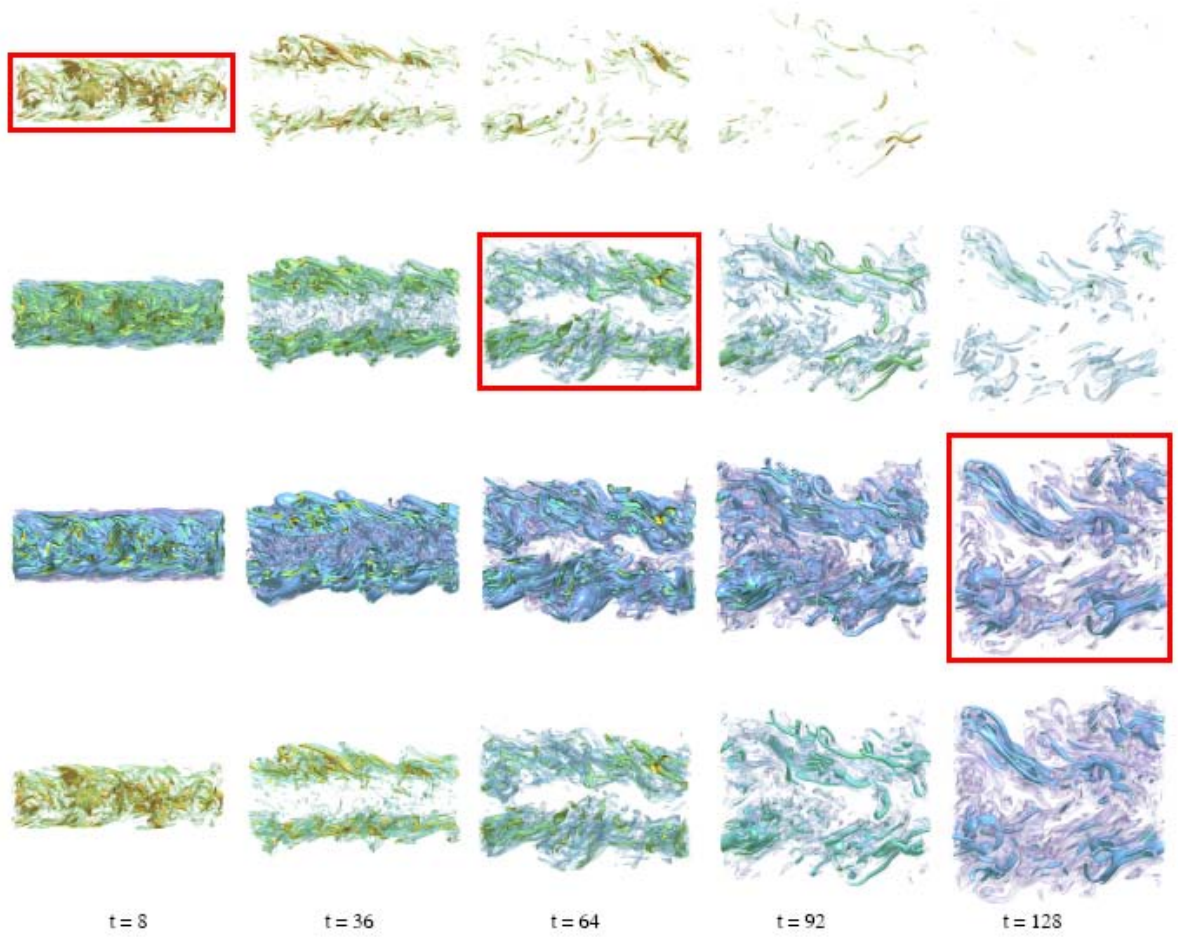


Figure 5: A DNS turbulent reacting plane jet data set using IATF. The key frames are bounded in red, and the transfer function used for each key frame is also applied to different time steps shown in the same row. The fourth row shows the rendered results using the derived transfer function. Taken From F.-Y. Tzeng and K.-L. Ma, "Intelligent Feature Extraaction and Tracking for Large-Scale 4D Flow Simulations," [13]

greatly simplified and intuitive visualization interface.

In volume rendering, a one-dimensional transfer function is a simple ramp, piecewise linear or arbitrary table function that maps between the original data to an opacity and a color to be rendered within a given computational volume element (voxel). Through this one-dimensional mapping, transfer function, one can enhance the visibility of desired spatial structures within a given visualization as will be shown in following sections.

In the IATF, the user manually adjusts a one-dimensional transfer function for selected time steps to highlight features/regions of interest, and then an adaptive transfer function for the whole time sequence is automatically generated using the machine learning system. In this case, the adaptive transfer function takes into account the scalar field and the data distribution variations over time. The tracked feature can be represented by any type of graphical object. One such object they used was a volume rendered feature tracking technique based on such intelligent transfer functions. The user manually highlights features during rendering as the tracking proceeds, and enables a variety of highlighting criteria to enhance the features of interest.

Figure 5 illustrates Tzeng and Ma's approach. The user first selects a couple of key frames and assigns a 1D transfer function for each key frame to define the features of interest. These 1D transfer functions are then sent to a machine learning engine for training. Training is an iterative process and the trained machine learning engine is able to generate an adaptive transfer function by using these image driven transfer functions, as well as the data-driven properties of the data set. During rendering, the adaptive transfer function is used to assign opacity to each voxel. The user can visualize the rendered results using the adaptive transfer function and add new key frames when needed.

2.5. FieldView and GUI for Volume Rendering, PCFET and IATF

FieldView version 13 was used as the baseline platform for the prototype where by Volume Rendering, Python wrapped servers and a QT based prototype GUI to control the data extracts and Volume Rendering were added. This prototype version was designed to support polygonal surfaces (i.e. boundary surfaces, iso-surfaces, computational surfaces) displayed along with volume rendered images, client side (desktop/workstation) Volume Rendering so as to exploit the local GPU.

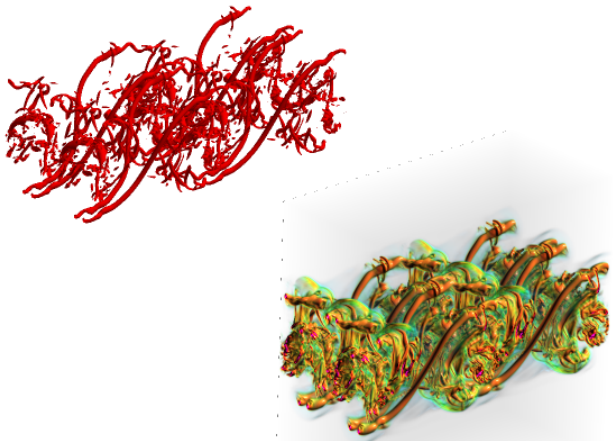


Figure 6 - Iso-Surfaces and Volume Rendering of a TML

IFDT utilizes a client-server paradigm whereby the FieldView client machine communicates via secure sockets to the master server and the FieldView master server in turn communicates via MPI with the slave servers. Each slave server machine performs operations on a single grid.

A ray traced volume render shader code was added to the FieldView 13 scene graph. For volume rendering, the volume data is stored into a 3d texture. The sides of the volume cube are sent as normal polygons. The shader is presented with each pixel of each side of the cube. From this pixel and the eye location, it draws a ray into the volume and computes the pixels' color. At this time, only regular Cartesian grids are supported which requires a one to one correspondence between voxels and

resampled grid elements. Currently, the code does not correctly render mixed volume and polygonal data; the polygonal data renders behind the volume data. Figure 6 shows an image of a volume rendered region (lower-right image) displayed together with polygonal data surfaces (upper-left image) generated for a result from LESLIE3D of a Temporal Mixing Layer.

2.6. IFDT Panel Implementation

The Predictor Corrector Feature Extractor and Tracking (PCFET) graphical user interface from Ma et.al. [14] was integrated into FieldView and coupled with LESLIE3D. Figure 1 illustrates FieldView version 13 with the PCFET environment and a LESLIE3D case running in-situ with FieldView. The image shows both the Volume Rendered vortex on the lower right and polygonal rendered iso-surface of vorticity magnitude on the upper left corner. The method works either in direct mode whereby all the tools execute on one desktop system or in a client-server mode whereby the solver and Predictor-Corrector method execute on a remote server while the Volume Rendering and User Interface run on the user's local client. Three new panels have been integrated into FieldView. The top left panel (IFDT), is the main control panel for IFDT which allows the user to select when to cause FieldView to pause the flow solver, copy current data to memory, release the solver to continue solving and then render a polygonal image of the solution at the copied time instance. Full FieldView functionality is maintained such that the user can create iso-surfaces, 2-D plots or point-probe as if using the standard FieldView product.

Figure 7 shows the "Control Center Panel" for the IATF and the Predictor-Corrector Feature Extraction. The "IATF" tab panel contains the transfer function histogram which the user can manually manipulate. In the example shown, the transfer function color map used the same color map and scalar function of Vorticity Magnitude as the FieldView "NASA-1" colormap and was adjusted to display the vortices. The IATF panel has a push button on the transfer function GUI to create default Gaussian curves which make data exploration easier. A Gaussian curve can be created and moved left or right to search for features. This operation is somewhat similar to sweeping an iso-surface in FieldView. However, unlike an iso-surface the Gaussian can span a range of values and since the resample volume data resides in memory, when the position of the Gaussian changes the image immediately updates.

Lighting parameters can be directly controlled on the GUI. These controls include the Ambient Lighting fraction, Ka, the Diffuse lighting fraction, Kd, Specular fraction, Ks, and Shininess. In addition, "Sampling" controls the number of steps taken along the ray cast. By moving the slider to the right, the ray cast results in less steps, coarser images but faster translation and rotation of the graphic object. If the user moves the slider to the left, this action would result in a greater number of steps along the ray cast, much better resolved images but slower response.

The "Track" Tab controls the tracking and extraction of the flow features. When the user presses "Start" on the Track tab the system will begin to record volume data snapshots on the client. The system continuously performs the volume rendering steps as described above. At every N-th time step as requested by the user, the system collects statistics information of the tracked feature. To avoid reducing solver performance a request to the solver for volume data is made at a different user defined time interval. This recording of volume data values for rendering will continue until the stop button is pressed or if the record-buffer fills, currently defined as 25% the size of physical RAM. The IATF tab contains a "Time Steps" slider. Moving the slider to any position will display that recorded volume data. This slider always has a maximum value equal to the number of captured volumes.

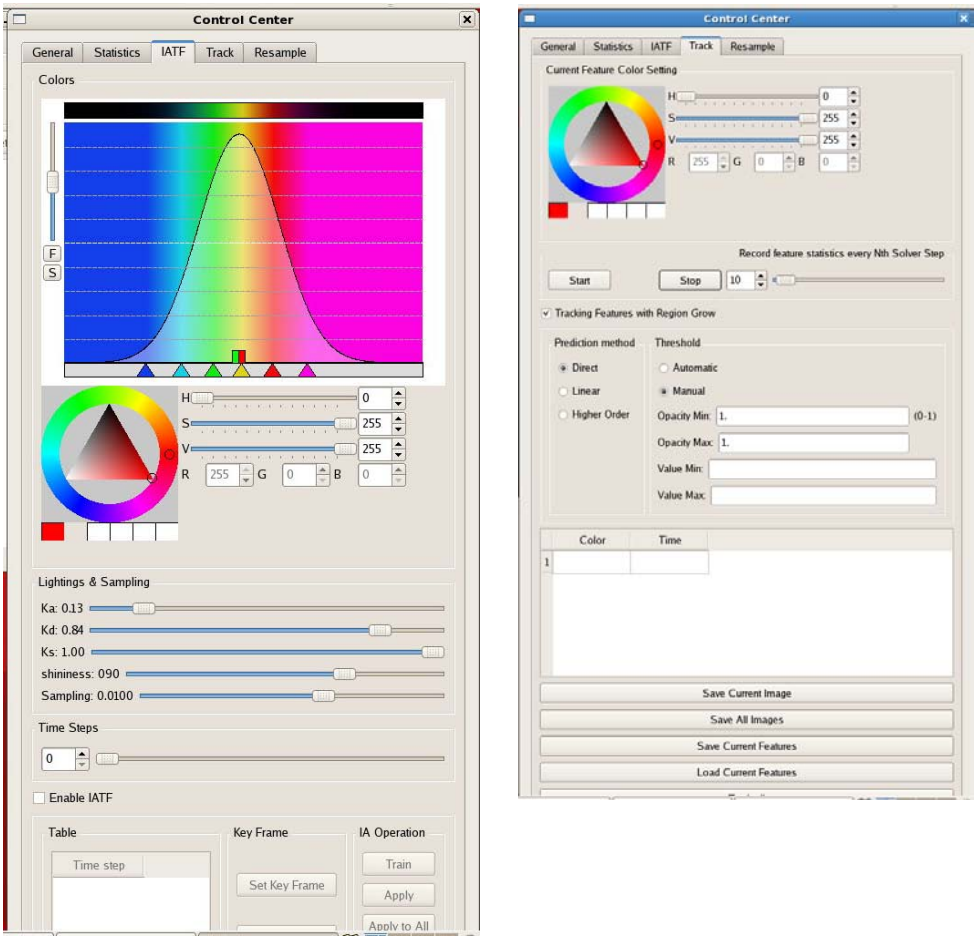


Figure 7 - IATF and Track Control Panel

The Track tab panel also controls the "Picking" and "Extraction" of a tracked feature. There are two settings for "Threshold" which controls the range of values that shall be tracked and extracted – Opacity and Value. The "Value Min" and "Value Max" set the minimum and maximum bound of the voxels that shall be tracked, respectively. The "Opacity Min" and "Opacity Max" set the minimum and maximum value of those voxels that shall be included in the tracked feature. Once these values have been set, the user may then "Pick" a feature for tracking by manually clicking a desired feature displayed in the volume rendering. The voxels of the feature of interest whose scalar value is bounded by the two sets of threshold min max are then colored by a single color that the user selects with the color wheel and triangle shown in Figure 8; currently set to red. As an example, Figure 8 shows a resulting volume rendered image displayed in the FieldView graphics window which shows the resulting image of the vortex flow (colored voxels) and the picked voxels shown in

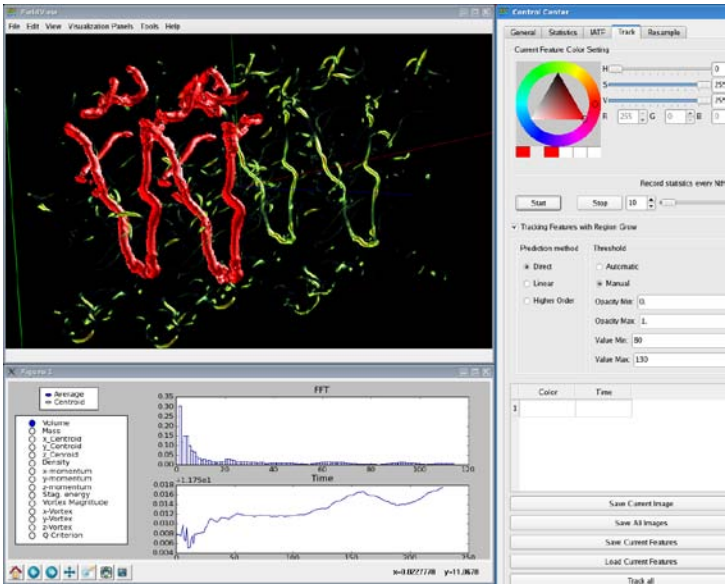


Figure 8 - Control Panel – Track Tab Panel Showing Tracked Feature in Red

red.

As the system collects time history data for the picked and extracted feature, the “Statistics Viewer”, panel in the lower left of Figure 8, allows the user to interactively select different scalar quantities for monitoring and whether that data is either an average of that quantity over the entire feature voxel set or the value of the scalar at the centroid of the picked feature voxels.

Figure 9 illustrates the training process for IATF as applied to a temporal mixing layer as simulated by the LESLIE3D code. For the example presented here, the grid size is 65 x 65 x 33 grid points. The volume rendered solution is colored by Q-Criterion and the transfer function was manually set by the user. The colormap was set to be consistent with FieldView’s NASA-1 colormap and the opacity shape (the black S-shape line or sigmoid) is known through experience to highlight the vortex core.

As shown in Figure 9, the IATF panel contains the transfer function currently being used to create the volume rendered image. The “Time Steps” slider allows the user to interactively move to any of the re-sampled time steps that were loaded into memory during the previous “tracking” step. The user can move the slider to any time step, manually adjust the transfer function and interactively view how that transfer function affects the volume rendered image.

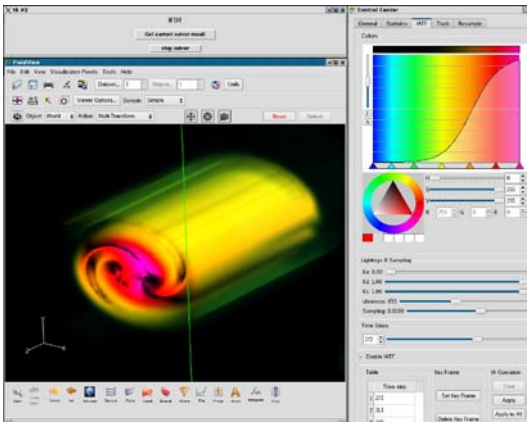


Figure 9: Temporal Mixing Layer with Derived Transfer Function Applied & IATF Panel Shown

The user creates the set of training data for the derived transfer function by repeatedly advancing the Time Step to advancing positions, manually adjusting the Transfer Function to highlight the feature and then click on “Set the Key Frame”. In the example shown in Figure 9, the user has moved the slider to Step 272, interactively adjusted the transfer function to highlight the flow feature and then selected “Set Key Frame”. They repeated the above steps for time steps 313 and 395 as evidenced by the “Time Step” Table.

Once the user is satisfied with the set of training data, they then would click on “IA Operation -> Train” which causes the IATF to compute derived transfer functions based upon the Keyframes. Once the training is complete, the derived Transfer Functions may be applied to either select time steps or to all the saved time steps. For the example shown in Figure 9, we see the derived transfer function being applied to Time Step 272.

2.7. Transfer Function Usage for Feature Discrimination

In volume rendering, a “Transfer Function” controls the color and opacity displayed in the graphic. In the IFDT user interface, the Volume Rendering user interface is controlled in the Control Panel - IATF Tab. Figure 10 shows the Transfer Function user interface (UI). The X-axis represents the scalar function value of currently displayed Time Step from the minimum value on the left to the maximum value on the right. The user can select the different colors along that spectrum by selecting colors from the color wheel in the IATF tab and clicking on the triangles shown. The triangles can slide to any position hence controlling the position of the scalar color.

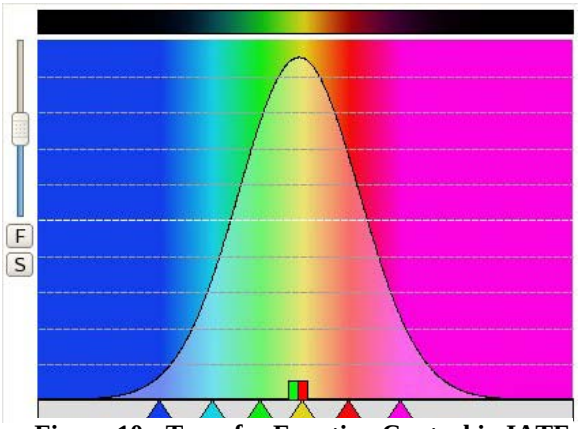


Figure 10 - Transfer Function Control in IATF Panel Tab

The y-axis of the Transfer Function User Interface represents the opacity. The user controls the opacity by manually scribing a line via mouse controls in the colored area. A line at the bottom corresponds to completely transparent while a line at the top of the vertical scale represents fully opaque. The color spectrum stripe at the top represents the color that would be shown in the graphic.

The example shown in Figure 10 uses the Gaussian tool; enabled by clicking on the "F" button. The user can control the width of the Gaussian by moving the mouse up and down while simultaneously click and holding on the red rectangle. Similarly, the user controls the height of the curve by click and holding on the green rectangle and moving the mouse up and down. The horizontal position of the Gaussian can be moved to the left or right by click and holding on the green rectangle and moving the mouse to the left or right.

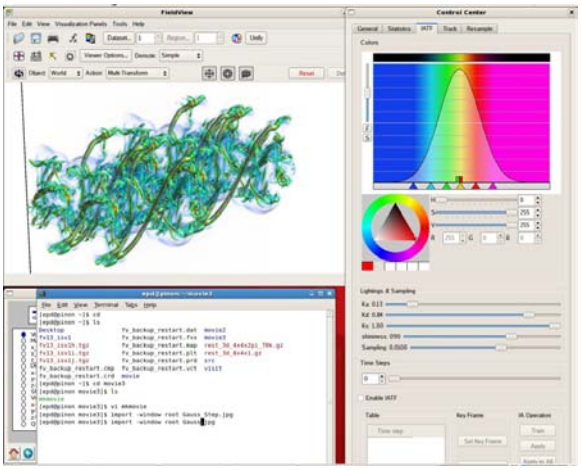


Figure 11 -Gaussian Transfer Function

The Transfer Function UI is a very useful tool for interactively highlighting and evaluating flow features - Feature Discrimination. There are four types of transfer function shapes that may be applied by the user.

1. Gaussian
2. Step function
3. Sigmoid – "S-Curve"
4. Delta Function (Sharp Gaussian)

Figure 11 shows the Gaussian function and the resulting image. As shown, the color at the peak of the curve causes the voxels corresponding to that scalar value to be colored accordingly and as a solid otherwise known as opaque). The voxels corresponding to scalar values to the left and right shall be assigned different colors and less opaque (transparent).

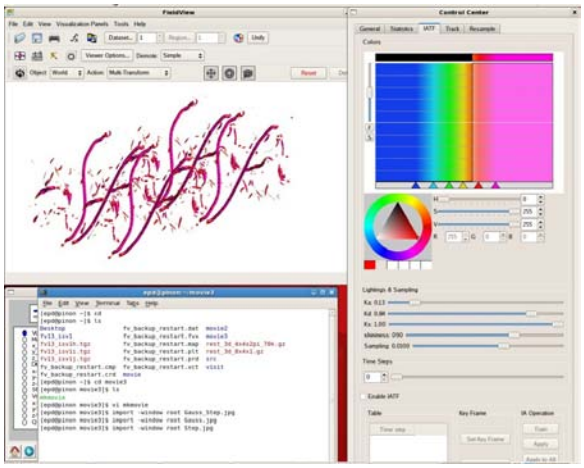


Figure 12 : Step Function Transfer Function

A Step Function as shown in Figure 12 is another transfer function type. To the left of the step all corresponding voxels shall be completely transparent. Voxels to the right of the function are completely opaque. This function is manually drawn using the mouse by click and holding towards the top of the color spectrum window and sliding the mouse. The Step Function can be used to highlight features such as the vortex structures shown. By moving the step to the left or to the right different features will show with no extraneous transparent voxels; needed for good feature picking.

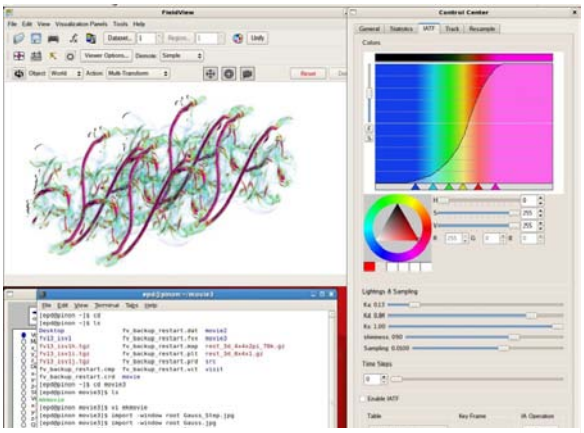


Figure 13- Sigmoid (S-Curve) Manual Drawn Transfer Function

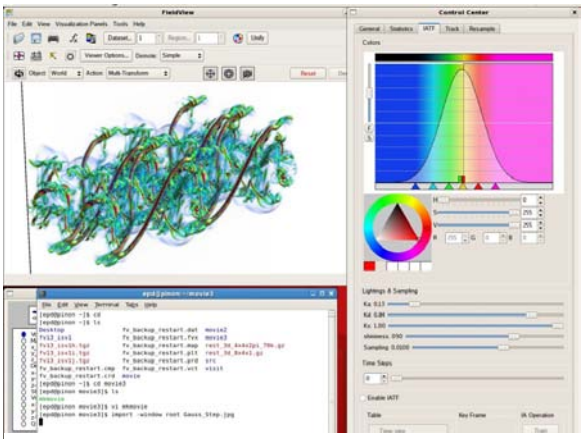


Figure 14 - Gaussian + Step Transfer Function

The Sigmoid or “S-Curve”, Figure 13, is either a hand drawn curve or automatically created by the IATF training algorithm. The S-Curve is a good method for both highlighting a feature and showing the flow around the feature. As shown here, the voxels in the magenta region scalar values are all drawn as opaque, while all other voxels of lesser value are less opaque smoothly fading out in the blue spectrum.

Combining the Gaussian and Step function as shown in Figure 14 is a good method for creating an S-Curve without hand drawing. The issue with the hand drawn S-Curve is that it requires care and manual mouse dexterity to get a smooth curve. By combining the Gaussian and Step function, two curves that have a more easily controled drawing UI, the user can control an S-curve much more readily. This combination can result in stunning graphic images as shown.

Any of the transfer functions presented so far can be used to discriminate flow features. For vortical flow features where maxima of either vorticity magnitude or Q-criterion are indicators for a vortex, it was found that a “Delta Function” that is approximated by a very narrow Gaussian curve is very useful for interactive exploration of the flow field. Figure 15 shows the application of the Delta function. As shown, the Gaussian is very sharp and narrow. The user can readily click and hold the Gaussian and slide it to the left or right and immediately see the effect upon the volume rendered image. This technique is very useful in finding the flow features of interest so that they can be picked for feature tracking and extraction.

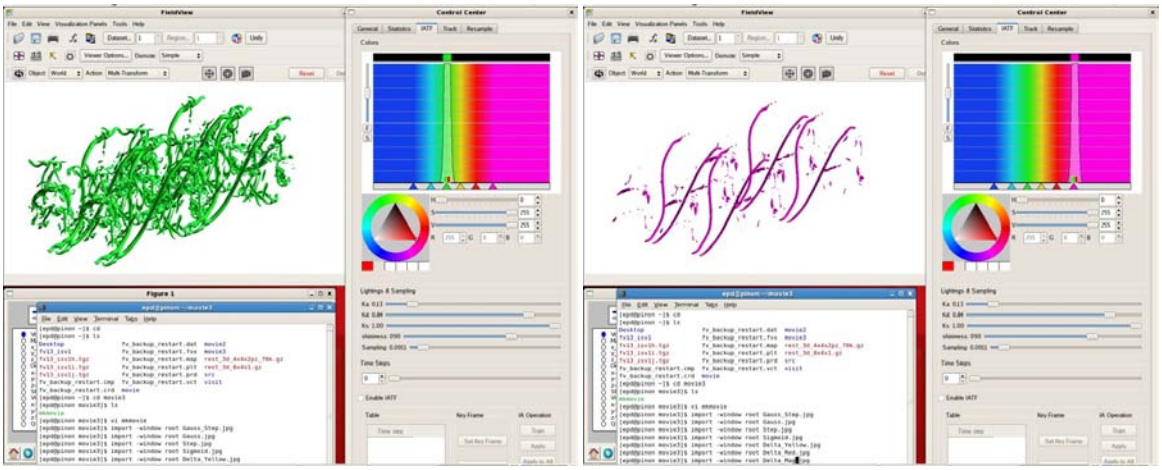


Figure 15 : Delta Function in Green and Magenta

2.8. Statistics Viewer

During feature tracking, transient information of the tracked feature is collected by the server processes and sent via a secured socket to the client for display in an interactive Statistics Viewer Panel as shown in Figure 16. Currently, time history for the feature volume, mass, position, density, momentum, stagnation energy, vorticity magnitude, vortex position and Q-criterion is collected for all the voxels within a given tracked feature. The Average value of all the voxels contained in the feature and also the value at the Centroid of the volume of voxels is also determined and collected. The user can then select to display either the Average or Centroid value of any of the feature values collected.

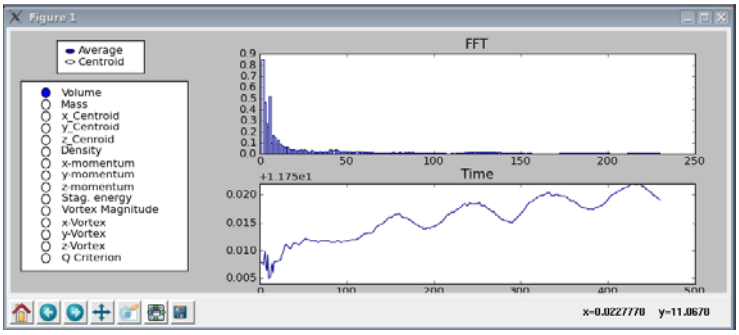


Figure 16 - Statistics Viewer Panel

The statistical data is displayed as either an FFT (2-D plot on the top) or as a time history (2-D plot shown on the bottom). For the example shown, the FFT displays the Fourier Modes on the x-axis and the magnitude on the y-axis for the Volume of all the Voxels in the picked feature. In the time plot, we see the time evolution of the feature with time on the x-axis (time steps) and the total feature volume on the y-axis.

3. Results

3.1. Test Case – Temporal Mixing Layer

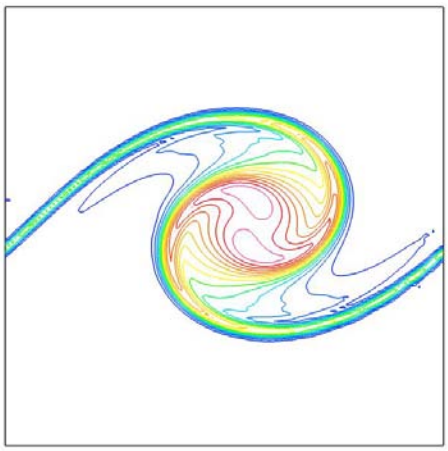


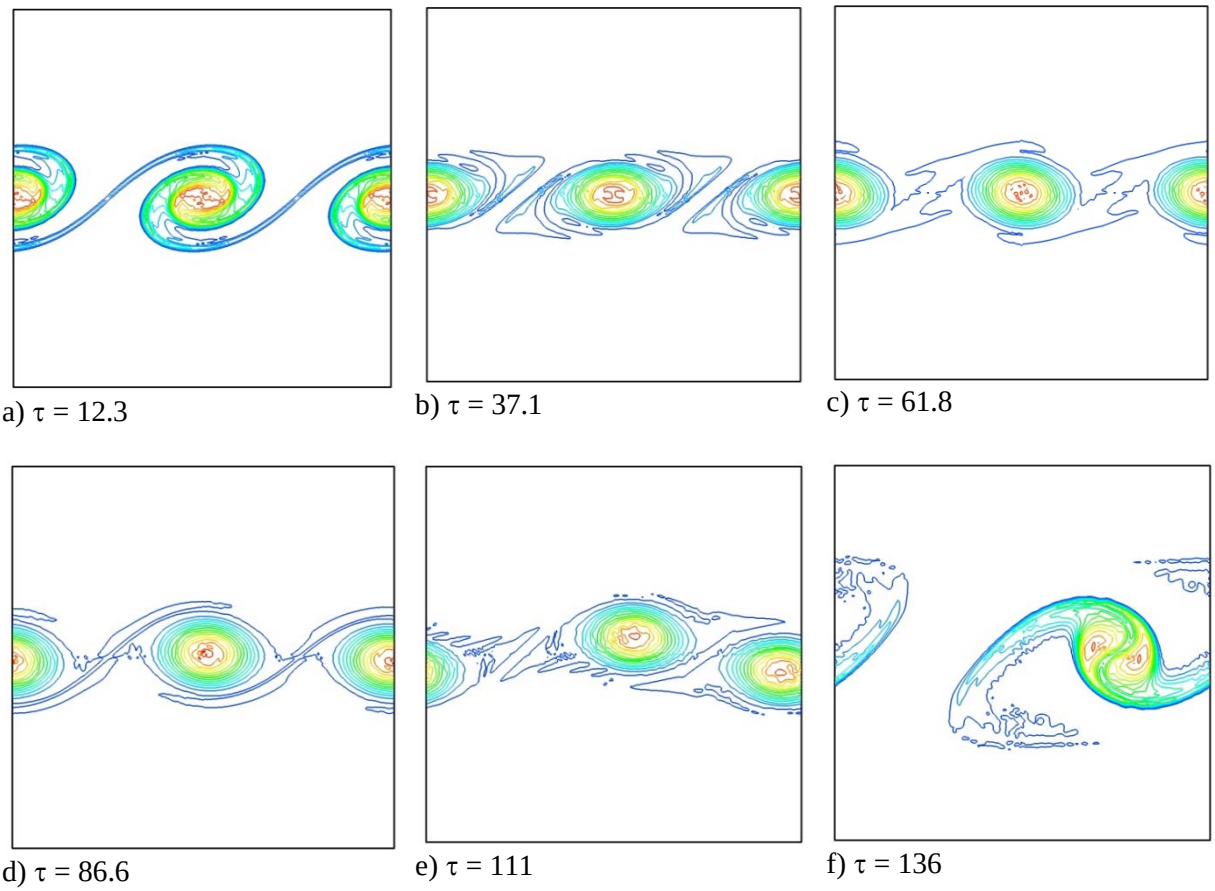
Figure 17 - . 2-d vortex roll-up in planar temporal mixing layer with (2x2)π domain. Vorticity magnitude is plotted

The test problem simulates the unsteady dynamics of a *temporally evolving planar mixing layer* (TML). This type of fundamental flow mimics the dynamics encountered when two fluid layers slide past one another and is found in atmospheric and ocean fluid dynamics as well as combustion and chemical processing. The two sliding fluid layers are subject to inviscid instabilities and can evolve from largely 2-d laminar flow into fully developed, 3-d turbulent flow. See Metcalfe et.al. [15] and references therein for further mathematical details on the subject.

The initial velocity profile of the LESLIE3D TML benchmark leads to vortex roll-up and, depending on the domain extent, vortex pairing due to the specially tuned 2-d instabilities introduced in the initial profiles. An example of the 2-d vortex roll-up is shown Figure 17, whereby the 2-d flow has evolved from the initial unstable velocity profile into a classical roll-up shape and is dominated by the single span-wise vortex over the (2x2)π domain. The TML tracks this flow feature in time and has no mean motion.

A larger 2-d span-wise domain length allows multiple span-wise vortices to roll-up. These vortices can, in turn, pair and merge into a single vortex. A time sequence of this phenomenon is shown in the

following image sequence. The time sequence simulation uses a (4x4)π domain with 256x256 grid resolution. The first image, Figure 18(a), shows a close resemblance to the single-mode result shown in Figure 17. The two vortices remain largely stationary; however, they mutually induce a small relative velocity. Eventually, the small perturbation grows causing a rapid pairing of the two vortices. Note, the peak vorticity is constant during the entire simulation. Viscosity and numerical dissipation slowly dissipate energy causing the only change in vorticity.



a) τ = 12.3 b) τ = 37.1 c) τ = 61.8

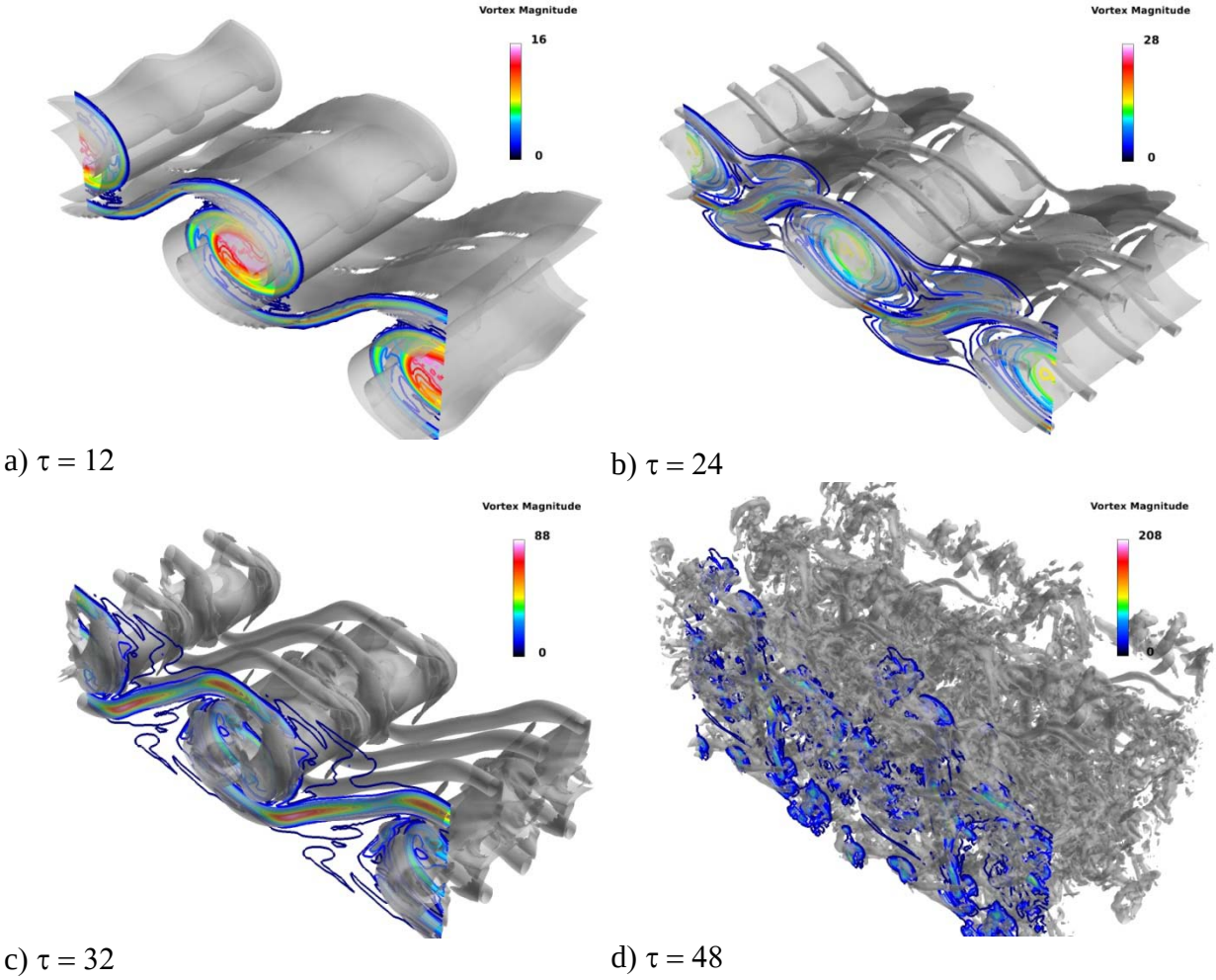
d) τ = 86.6 e) τ = 111 f) τ = 136

Figure 18. Time sequence of 2-d TML over (4x4)π domain. Showing vorticity magnitude with maximum at 15 s-1.

Three-dimensional instabilities can also be introduced to the shear layer. Under these conditions, counter-rotating vortices develop in the highly strained region between the dominate 2-d span-wise *rollers*. These *rib* vortices can inhibit 2-d vortex pairing if sufficiently energetic and are dependent upon the balance between the instability energy growth and dissipation. These rib flow structures enhance the mixing rate between the two fluid streams due to the smaller size and high vorticity.

An example of the unsteady 3-d vortex dynamics in the TML is shown in the following sequence of 4 images. These images show the vorticity field at equally spaced intervals for a TML computation initialized with both 2- and 3-d perturbations. The computational domain is (4x4x2)π with a uniform mesh of 256x256x128. The first image, at non-dimensional time τ = 12, shows the iso-scalar surface of vorticity magnitude at 20% of the peak value is shown along with a contour plot of the vorticity magnitude. At this early point in the TML evolution, the vorticity is still largely confined to a nominally 2-d vortex sheet and the peak vorticity (18 s^{-1}) is not significantly above the initial

distribution. However, the 3-d perturbations are evident in the span-wise direction. The 2nd image, at τ = 24, clearly shows the growth of the 2 counter-rotating rib vortex pairs in the high strain regions. Again, the iso-scalar surface is at 20% of the peak which has increased, due to non-linear vortex dynamics, to 28 s^{-1}. The increase in peak vorticity is confined to the high-strain "braid zone". The vorticity originally distributed along the span-wise vortex sheet in the braid zone is rolling into stream-wise ribs. Recall that the peak vorticity is constant throughout the 2-d time sequence shown in Figure 18; non-linear vortex stretching is only present in 3-d flows. The 3rd image in the sequence, at τ = 32, again shows the rapid increase in the intensity of the rib vortices as they are further stretched. The final image, at τ = 48, shows a highly turbulent flow-field. The intense rib vortices have penetrated the 2-d span-wise rollers and destroyed the coherence of the TML. Unlike the 2-d TML, pairing of the 2-d span-wise rollers was suppressed by the influence of the 3-d perturbations.



a) τ = 12 b) τ = 24

c) τ = 32 d) τ = 48

Figure 19. Time sequence of 3-d TML on (4x4)π domain with 3-d perturbations.

3.2. Interactive use of IFDT

The following presents a step by step use of IFDT to track a picked feature simulated by the Three-Dimensional TML case. The IFDT system is started from the command line by first initializing an "mpd-ring" on the remote server where LESLIE3D and FieldView servers execute and the starts the statistics viewer on the client which communicates to the server side across a secure socket connection. The IFDT system then sequentially initiates an *ssh* remote call to the server to launch the SIF, launch LESLIE3D using *mpiexec*, and open another secure socket connection across which the

FieldView client process communicates to the FieldView master server. FieldView and the IFDT panels will then show as illustrated back in Figure 1.

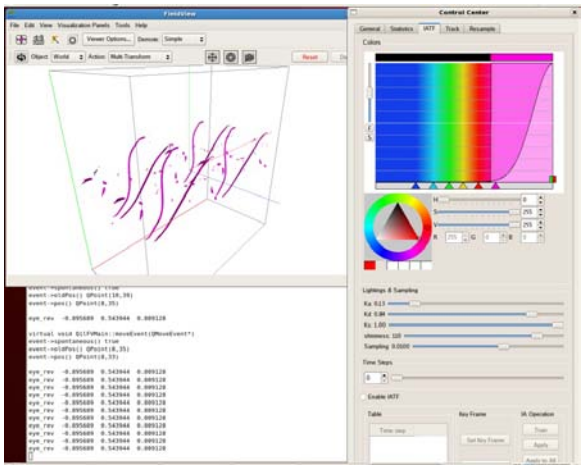


Figure 20: IATF Tab Panel, Transfer Function with Gausian and Step Function

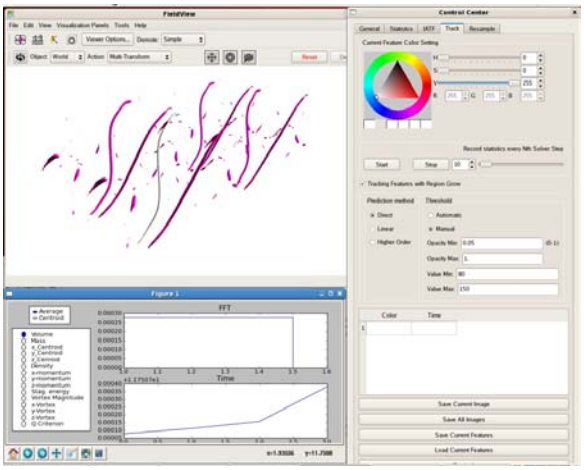


Figure 21: Feature Picked and Tracking

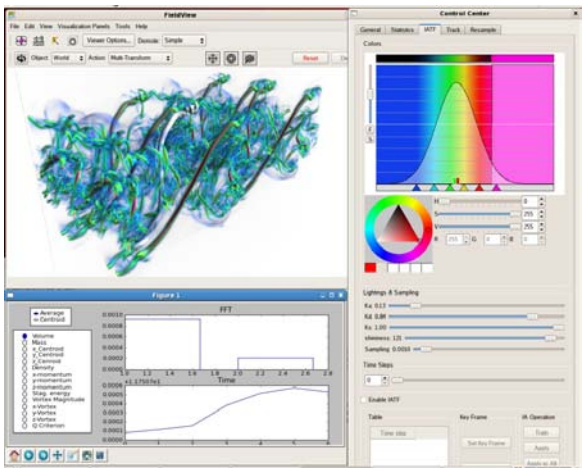


Figure 22: Combined use of a Step and Gaussian Transfer Function

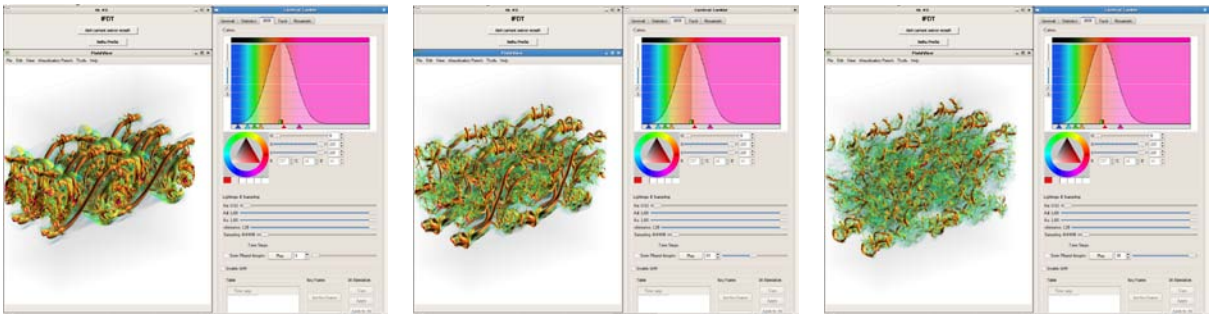
Next, the user would start the FieldView slave servers and load the latest solution data by clicking on the IFDT GUI button “Get current solver result”. This action initiates a Manual Start FieldView client server process, causees the SIF to pause the LESLIE3D process and the FieldView slave servers to load the current data. Once the FieldView slave servers have loaded the current data, the flow solver is released and it continues to solve. FieldView then executes a corresponding FieldView restart which creates a surface and creates an image. At this point, the user has full control of FieldView and can create surfaces as in standard FieldView.

For a feature pick operation, IFDT simply grabs the first non-zero opacity voxel along the line of sight where the pick occurred. Therefore, the best transfer function for picking and discriminating features is either a delta function or step function as shown in Figure 20. To pick, the user first adjusts the transfer function until the desired feature appears on the screen. The user will then click on the “Track” panel and click on the “Tracking Features and Region Grow” radio button. The user would then adjust the picking color through the color wheel User Interface. The user activates the pick color by clicking the left mouse button on the left square below the color wheel; “White” in this example. The threshold range sets the scalar function value range for a given feature. By clicking on Manual, the user manually sets these threshold ranges. The user would then hold down the “Shift” button on the keyboard and place the mouse select arrow over the desired feature and click the left mouse button on the feature. If a successful pick occurred, the client will emit a “beep” sound, write the voxel location to the console window of the pick and the feature will become colored as shown in Figure 21. In this example, one of the vortex braids has been highlighted with the color white.

To start the feature tracking, extraction and statistics gathering, the user would enter under the “record statistics every Nth solver step a value for

data sampling. The valid range is from 1 to 500. When the user clicks on “Start” the feature tracking begins. At every Nth solver step, the feature statistics information is computed and the statistics information is sent to the client and then displayed on the statistics viewer. The volume render graphic shown on the screen and the resampled data volume and feature tags are sent to the client at different time intervals to maintain good interactive performance for the user.

The time difference between the recorded volume time history and the statistics gather gives the user interactive capability while gathering in-situ data from the flow solver. For the case shown here, the volume rendered update has been set to 5 minutes while the statistics and IATF volume history data is update every 10 time steps (approximately every 1 minute of wall clock time). In between the graphic updates, the user can change the transfer functions, rotate, pan and zoom the volume render graphic and even create polygon surfaces. Figure 22 shows the transfer function has been changed while the statistics is being updated simultaneously.



a) Time Step Playback 1 b) Time Step Playback 19 c) Time Step Playback 38

Figure 23 : IATF Tab - Time Step Playback

The recorded volume data may be interactively viewed with the IATF tab Volume data playback. Figure 23 shows the IATF Tab at three different recorded Time Steps (1, 19 & 38). The user may move the slider to any time step, click through to any time step, or click on the Play button to sweep through the series of recorded Time Steps. The user may also change the Transfer Function to explore the effect upon each time step. **This ability to sweep through and interactively explore transient data is a new method that helps the user gain more knowledge about their data.**

3.3. Discussion of Problem Size and Scale-Up issues

A series of benchmarks were conducted to assess the performance and scalability of the prototype software. The first series of benchmarks was designed to assess the scalability and performance of IFDT in a remote client-server configuration. Two grid sizes were used in the LESLIE3D TML simulation: 257x129x33 and 257x129x65. Each test was run using Intelligent Light’s compute cluster (Nilchi) as the FieldView server and compute engine and a remote client connected over broadband internet. Nilchi has nine (9) HP BL460c nodes, two dual-core Xeon X5260 processors per node, 16GB of RAM per node, 72GB SCSI storage on each node with both GigE and Infiniband network fabric between the head node and the 8 compute nodes. The head node of Nilchi was used as the FieldView master server and the 8 compute nodes as the FieldView slave servers and the CFD calculators. The network speeds between the head node to the remote client ranged between 64-128 kbytes per second depending upon load. The simulations were run for 500 CFD time-steps before timings were collected. Timings were collected over 1000 CFD time-steps. The lag in starting the timings was to allow sufficient time for the FieldView IFDT GUI to be manually attached over the internet and for the volume rendering to be configured as needed.

The impact of volume rendering on the computational throughput was measured by varying the frequency that the CFD dataset is sampled and rendered. This is controlled in the IFDT GUI by adjusting the *Nth-step* value. The functionality of the Nth-step controller forces synchronous interaction with the CFD solver. Dataset sampling is initiated and completed synchronously every Nth-step of the CFD solver. Volume image update is still done asynchronously.

Volume sampling of the CFD dataset has two purposes in the IFDT. First, it is used in the feature tracking application. Simple time history statistics on the tracked features can be computed. A fixed sampling frequency is required in order for these computed statistics to be properly analyzed in spectral space leading to the change in meaning of the Nth-step control variable described above. Volume sampling is also used in the volume rendering phases on the IFDT client. Since this is only for visualization purposes, this operation is still allowed to be asynchronous. Note, the server-to-client network transmission requirements are different for these two features. For feature statistics, only the reduced values must be transmitted from the FieldView server to the remote client. This will be less than 1 kilobyte. Volume rendering requires that the entire re-sampled volume data be transmitted to the client and is dependent upon the re-sampled mesh size.

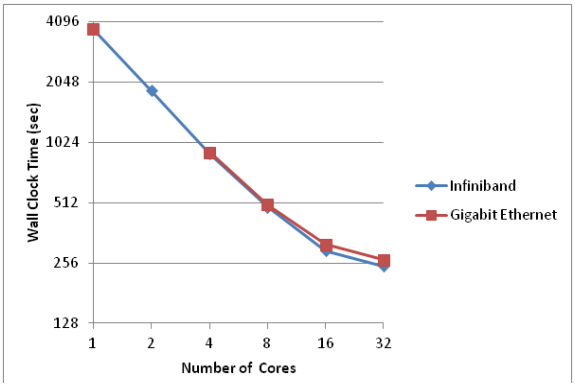


Figure 24: Total wallclock time (sec) vs. number of the cores for IFDT software for medium grid size (256x128x32) using IPoIB (blue) and GigE (red) networks.

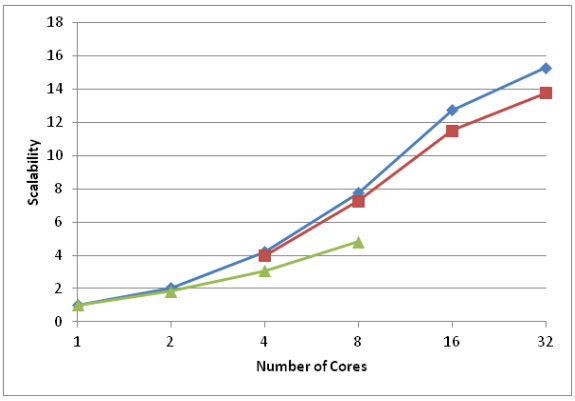


Figure 25: Scalability of the LESLIE3D solver using the medium (256x128x32) grid on the Nilchi test cluster using IPoIB (blue) and GigE (red) networks. Scalability on a shared-memory workstation (8-core) shown in green

The raw performance of the IFDT software without any re-sampling is shown in Figure 24 for the medium size grid (256x128x32). These performance measurements were obtained on the Nilchi test cluster using the high-speed Internet Protocol over the Infiniband (IPoIB) and the standard Gigabit Ethernet (GigE) networks. The IPoIB network provides close to 10Gigabit speed; however, it does not provide the low latency associated with some of the hardware-assisted protocols designed specifically for Infiniband.

With no re-sampling or any other interaction from the FieldView server, the Figure 24 graph shows the performance of the CFD solver LESLIE3D itself. The parallel scalability on the Nilchi system is shown in Figure 25 . The parallel efficiency drops from nearly 100% when using 1 core per node (8 cores total) to less than 50% when 4-cores per node are used (32 cores total) with both networks. The network speed appears to have a much lower impact on the scalability than the core loading. The scalability of the same benchmark was also measured on a shared-memory 8-core workstation (green line in Figure 25). No network fabric is required for this parallel computation but the parallel efficiency is still seen to be poor strengthening the argument that the inter-node network speed is only marginally important in this scenario. For this reason, only the faster IPoIB results will be discussed further.

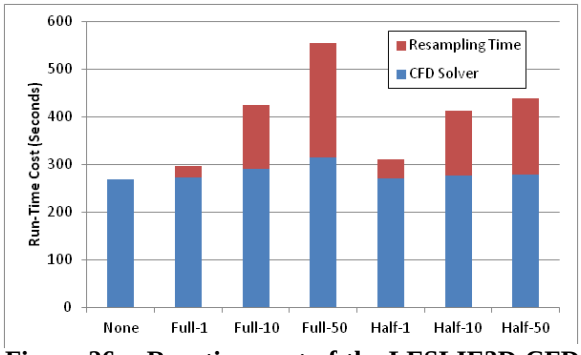


Figure 26: . Run-time cost of the LESLIE3D CFD solver and re-sampling algorithm as a function of re-sampling frequency (Nth-value) and re-sampled resolution (full vs. half). Using Large grid with 16 cores

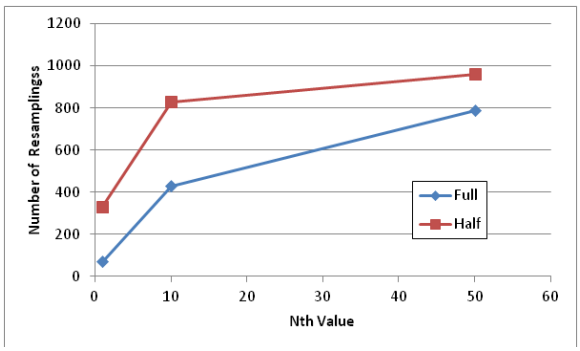


Figure 27: Total number of re-samplings as a function of Nth value over 1000 CFD time-steps for full and half resolution re-sampling volumes. Using Large grid with 16 cores.

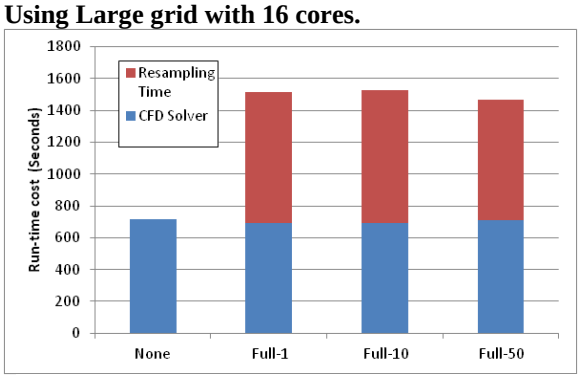


Figure 28: Run-time cost breakdown for shared-memory in-situ simulation on Medium grid with 4 cores

The next phase of the benchmarks measured the impact of in-situ processing on the total simulation run-time. Figure 26 shows the run-time breakdown of the in-situ simulation. The LESLIE3D large grid (256x128x64) was used with the IPoIB network on 16 cores. The cost of the LESLIE3D solver and the re-sampling components of the compute processes are shown. Two re-sampling resolutions were tested: full and half. At full resolution, the re-sampling mesh matches the underlying CFD grid. At half resolution, the re-sampled space is half the resolution in each direction for a total 1/8th the re-sampled grid size. The Nth-step value was varied from 0 (no re-sampling) to 50 and, in this context, the Nth-step value represents the frequency at which data is sent from the FieldView master to the client for volume rendering compared to the re-sampling rate. That is, when Nth is 10, the volume and feature-tracking data is sent to the client every 10 re-samples. This was described previously as *asynchronous* re-sampling.

Intuitively, a higher Nth value should result in a lower re-sampling cost. However, the opposite trend is seen in Figure 26. Instead, the re-sampling cost rises quickly as the Nth value is increased for the full resolution re-sampling case. The cause of this non-intuitive behavior is directly caused by the asynchronous re-sampling procedure. During asynchronous feature tracking, re-sampling is initiated immediately and only interrupted when data is transmitted to the client. That is, there is no correlation between the CFD solver time-step and the re-sampling frequency. Since the re-sampling cost is vastly cheaper than the transmission cost in the client-server configuration, *the effective re-sampling rate is greater for a higher Nth value*. This is clearly seen in the Figure 27 where the number of re-samplings is plotted against the Nth value for the full and half resolution cases. The greater number of re-samplings for the half resolution case is easily explained with the same logic: the cost to re-sample the volume is roughly 1/8th the full-resolution and the cost to transmit the data to the client is also approximately 1/8th meaning a higher number of re-samplings can occur within the same run-time. Note, however, that the total run-time cost of the in-situ simulation with the half-resolution is considerably lower despite the higher re-sampling frequency. Again, the transmission and re-sampling costs are 1/8th of the full-resolution reducing the in-situ overhead.

The impact of network bandwidth on the re-sampling frequency was measured by running a similar simulation on a shared-memory workstation. There, the FieldView client and the in-situ

processes were run on the same machine removing the need for network communication. Due to memory constraints, only the Medium grid was tested in this scenario. This simulation used 4 CPU cores for the CFD and re-sampling processes Figure 28 shows the same run-time breakdown of the CFD solver and re-sampling components for different Nth-step values. However, unlike the previous result, the re-sampling component has zero dependency upon the Nth-step value. In this scenario, 1000 re-samplings were executed over the 1000 CFD time-steps for all Nth-step values. Since there was no data transmission overhead, the asynchronous coupling allowed the re-sampler to continuously query the CFD solver for new data at the same rate the CFD solver could process a time-step. Aside from changing the frequency that the volume rendered image is refreshed, differing non-zero values of Nth-step have no impact on the run-time.

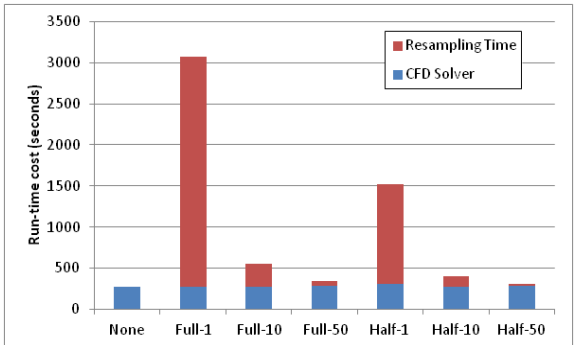


Figure 29: Run-time cost breakdown using synchronous Nth-step method on Large grid with 16 cores

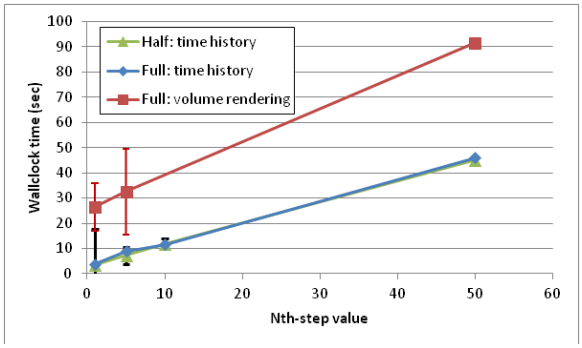


Figure 30: Client-side update time (sec) in client-server mode with 256x256x128 grid. Error bars represent percent deviation from mean

The Nth-step value can alternatively be used in a synchronous mode. In this scenario, re-sampling is done every Nth CFD time-step. Data is transmitted to the client and rendered at every Nth-step as well. This effectively synchronizes the CFD solver to the FieldView rendering client. That is, over a 1000 CFD time-steps, 20, 100 and 1000 re-samplings will occur for Nth values of 50, 10 and 1, respectively. The run-time costs of the re-sampling and CFD solver components of the in-situ simulation are shown for this scenario in Figure 29. The run-time for the Nth-step = 1 is seen to be substantially increased compared to that shown in the alternative scenario, Figure 29. As the Nth-step value increases from 1 to 50, the run-time quickly reduces until the cost is almost entirely attributed to the CFD solver, an ideal outcome for the end-user. This run-time scenario has the additional benefit of providing regular tracker feature histories allowing for proper statistical analysis.

Figure 30 shows the run-time between successive time-history and volume rendering (image) updates. Again, only a small amount of data is required for the time-history calculator so the transmission delay should be relatively minor. The Nth-step value was varied from 1 to 50 and the re-sampling volume was either full or half the resolution of the CFD simulation. And, the Nth-step value is configured synchronously in this situation so the frequency of re-sampling is in lock-step with the CFD solver. Because of the low bandwidth required by the time-history, the refresh rate follows closely to the CFD run-time. The average wall clock time per time-step without re-sampling is approximately 1s. With full resolution re-sampling at each time-step (Nth=1), the run-time cost increases to 3.5s due to re-sampling, nearly a 200% overhead. However, this impact quickly drops as the Nth value increases to 50. There, the re-sampling cost is hidden by the much larger CFD solver time.

The volume rendered image refresh rate is also shown in Figure 30 using full resolution re-sampling. For the same Nth-step value, the delay time between updates is seen to be many times higher. This is directly caused by the transmission time between the remote client and server.

Currently, all volume rendering must be conducted on the client in order to harness the rendering efficiency of the graphics pipeline; that is, GPU's. However, CPU-based volume rendering on the FieldView server side may, in fact, prove more efficient since the substantial transmission delay would be avoided. Other options include a remote desktop in which GPU acceleration is used on the FieldView server directly.

The FieldView server computational overhead is more variable than the memory footprint since the dominant cost, re-sampling, is directly linked to the re-sampling rate, that is, the Nth-step value. Without feature tracking or volume rendering enabled (i.e., no re-sampling), the FieldView server overhead is very nearly 0%. Benchmarks on LESLIE3D with Nth-step = 50 and with full re-sampling resolution showed the overhead to be just under 20% using 16 processors. With half the CFD resolution, this overhead was only 9%. It is important to note that LESLIE3D uses an explicit time-step algorithm with a low per-step cost. This low per-step cost can inflate the re-sampling overhead ratio. Codes using implicit time-step algorithms, which are generally several times more expensive per-step, will likely experience a much lower FieldView server overhead.

3 Conclusion and Future Work

IFDT demonstrates a new prototype capability for feature detection and extraction of turbulent flows. A flow solver, LESLIE3D, was demonstrated to work well In-Situ with the various other components needed to visualize, gather statistics and track a selected flow feature. A prototype Volume Rendering capability was integrated in FieldView and presents a new capability that can be integrated into the commercial product. The Intelligent Adaptive Transfer Function (IATF) method was integrated and tested. Although the User Interface presented new capability and techniques for exploring large scale unsteady data, more research and study is needed to yield a more robust and computationally efficient training capability. The feature tracking, based upon the Predictor-Corrector Feature Extraction and Tracking (PCFET) method demonstrated the capability to discriminate, pick and track turbulent flow features. The resulting feature extracts demonstrated the capability to reduce the size of unsteady data and enable interactive exploration of the data which enhances discovery and understanding of large unsteady CFD derived data. The prototype IFDT system presents a new capability that enables scientists and engineers to more readily explore large unsteady datasets. As a whole, the feasibility to couple flow solvers and other data analysis techniques such as the Statistics Viewer and the Volume Rendering enabled FieldView was clearly demonstrated.

4 Acknowledgements

This work was funded by a Phase II STTR from the Air Force Research Lab. The authors would like to thank Dr. Fariba Fahroo, Air Force Office of Scientific Research for her support. We would also like to thank Prof. Suresh Menon, Georgia Tech, for providing the LESLIE3D code and to Prof. Marilyn Smith for her review of the work.

References

[1] "Intelligent Light," [Online]. Available: www.ilight.com. [Accessed May 2012].

[2] R. Haimes, "pV3: A Distributed System for Large-Scale unsteady CFD Visualization," AIAA Paper 94-0321, 1994.

[3] D. S. Thompson, J. S. Nair, S. D. Machiraju, M. Jiang and Craciun, "Physics-Based feature

Mining for Large Data Exploration," *Computing in Science and Engineering,* vol. 4, no. 4, pp. 22-30, July 2002.

[4] S. Menon, C. Stone, H. Soo and H. Feiz, "Modeling Actiive Control Technologies using LES," in *41st AIAA Aerospace Sciences Meeting and Exhibit*, AIAA Paper No. 2003-0840, Reno, NV, 2003.

[5] C. Stone and S. Menon, "Open Loop Control of Combustion Instabilities in a Model Gas Turbine Combustor," *Journal of Turbulence,* vol. 4, no. 4, 2003.

[6] M. Masquelet, S. Menon, Y. Jin and R. Friedrick, "Simulation of Unsteady Combustion in a LOX-GH2 Fueled Rocket Engine," *Aerospace Science and Technology*, vol. 13, 2009.

[7] T. Smith and S. Menon, "The Structure of Premixed Flame in a Spatially Evolving Turbulent Flow," *Combustion Science and Technology,* vol. 119, 1996.

[8] "http://www.spec.org/cpu2006/," [Online].

[9] J. Sitaraman, A. Katz, B. Jayaraman, A. Wissink and V. Sankaran, "Evaluation of a Multi-Solver Paradigm for CFD using Unstructured and Structured Adaptive Cartesian Grids," in *46th AII Aerospace Sciences Conference*, AIAA-2008-0660, Reno, NV, 2008.

[10] P. G. Buning, R. J. Gomez and W. I. Scallion, "CFD Approaches for Simulation of Wind-Body Stage Separation," in *AIAA 22nd Applied Aerodynamics Conference*, AIAA-2004-4838, Providence, RI, 2004.

[11] C. Muelder and K.-L. Ma, "Interactive feature extraction and trakcing by utilizing region coherancy," in *Proceedings of 2009 IEEE Pacific Visualization Symposium*, 2009.

[12] R. Huang and K.-L. Ma, "Region Growing Based Technique for Volume Visualization," in *Proceedings of the 11th Pacific Conference on Computer Graphics and Applications*, Washtington, DC, USA, 2003.

[13] F.-Y. Tzeng and K.-L. Ma, "Intelligent Feature Extraaction and Tracking for Large-Scale 4D Flow Simulations," in *SuperComputing 2005*.

[14] K.-L. Ma and J. Wei, *Personal Communication,* 2010.

[15] R. Metcalfe, S. Orszag, M. Brachet, S. Menon and J. Riley, "Secondary instabilities of a temporally growing mixing layer," *Journal of Fluid Mechanics*, vol. 184, 2987.