A Tractable Approach to Understanding the Results from Large-Scale 3D Transient Simulations

Robert Haimes*
Department of Aeronautics and Astronautics
Massachusetts Institute of Technology
haimes@orville.mit.edu
and
Kirk E. Jordan†
IBM T. J. Watson Research Center
kjordan@us.ibm.com

The data generated from large-scale scientific simulations, like the Department Of Energy's ASCI (Accelerated Strategic Computing Initiative) problems or NASA's HPCC (High Performance Computing & Communication) grand challenges, can easily and quickly overwhelm any storage resources available. This data size issue inflicts a huge burden on those tasked in assisting the scientists, engineers and designers in understanding the results from these simulations.

Following the traditional post-processing visualization approach it becomes clear that the computer, graphics and disk requirements point one to the acquisition of a large visualization workstation with terabytes of striped disk arrays. Additional effort is required in compressing the data so that a single calculation can reside on this visualization machine. The end result can limit the interactive nature of the visualization when so much data needs to be pulled off of disk and then expanded.

Often with a transient problem, sub-sampling (in time) is performed. If the algorithm used for post-processing requires small time-steps between snap-shots (e.g. particle traces) then the results can be misleading due to inaccuracy or simply fail due to numerical divergence. Also, a transition from one spatial topology to another (often an event of great importance) can easily be missed. Another, and significant problem with traditional post-processing is that the results do not answer the investigators questions directly, but require inference.

Another option, the one suggested in this paper, is not to perform traditional scientific visualization on the data but feature extraction. If designed properly, the results can directly answer the scientist's or engineer's questions and reduce the data to be stored by as much as five orders of magnitude. When this is coupled directly with the simulation (co-processing) the overall size of the output data set with temporal sub-sampling will be able to reside on a normal workstation. Plus, the entire resultant feature extracted dataset will probably fit on the large visualization machine for playback with high temporal fidelity.

Nomenclature

a	speed of sound
E	total energy
H	enthalpy
γ	specific heat ratio
λ	eigen value
M	Mach Number
p	pressure
ρ	density
$\vec{u}$	velocity
$\underline{V}$	velocity gardient tensor ($\nabla\vec{u}$)

*Principal Research Engineer
†Senior Program Manager

Introduction

Large-scale simulations of physical phenomena on high performance computing systems (often on massively parallel computing machines) are generating a wealth of data. This poses a great challenge when it comes to closing the loop: the task of analyzing the results. The knowledge gained can then be applied perhaps to modify design parameters or investigate alternative approaches appropriate to the problem under study. One key aspect of this kind of investigation is to understand the existence, evolution (or lack there of) regarding certain features often specific to the application domain. In the area of Computational Fluid Dynamics (CFD) for aerospace applications, features such as vortices, separation, and shocks are often of interest to the aerodynamisist and consequently are the objects sought after in many large-scale 3D time evolutionary CFD simulations. Domain experts in conjunction with computational scientists devise numerical schemes to solve complex coupled systems, why should one not expect a similar effort with the analysis and visualization of the results from these simulations?

Simulation of a suite of PDEs

Many of the problems of interest start out with the description of the physics in the form of Partial Differential Equations (PDEs). These equations may be recast as ordinary differential equations or other appropriate discrete approximations distributed on a spatial grid that represents the region of interest. This system can then be moved in time by invoking an integration scheme, usually by taking small increments. The size (and form) of the spatial discretization and the type of time integration is often problem specific. These factors depend on both the physics and the numerics applied in order to satisfy certain stability and accuracy constraints. The system of equations usually requires just a 'handful' of parameters, appropriate boundary and initial conditions for the region of interest.

Starting with this small set of parameters, the amount of resultant data can become enormous as the simulation evolves. The huge number of time steps required (to satisfy limiting the errors associated with the numerics) necessitates capturing information NOT at every time step in order to keep storage requirements down, i.e., sub-sampling in time. The data requirements are further increased when the objects of interest are the interactions between either different parts of the physical domain possibly involving different length scales (leading to simulations involving multi-scales) or different physics interacting, e.g. structures and fluids (leading to multi-physics).

Resultant data sets

The result of the various approximations and increased resolution can lead to huge I/O problems in order to save all the data for later processing. How to get the data off a large parallel machine and save it both in a place that is accessible and in a form that is useable can be a major issue. (Not to mention transforming what is a compute bound problem into an I/O bound problem.) This is further complicated in that the simulations can often create data faster than it can be stored and the storage space is often insufficient and limited. Now, add to this mix the fact that massively parallel computers are often remotely located therefore requiring network transfers of vast quantities of data. The I/O problem only further complicates our ability to gain knowledge from the simulations.

Historically, the data set has been the dividing line between the solver and the visualization system. This has allowed the solvers and visualization tools to be developed without interference. The negative side effect has been that the visualization community has lost the requirement for knowing what the data represents. The physics is removed if one views the data as just a collection of scalar and vector fields or even more abstractly as merely an array of data.

Customized solvers – generic visualization

To process this wealth of data, visualization libraries and toolkits have been produced. The idea behind much of this effort is to create a general-purpose visualization suite. Most systems provide the ability to cut through the volume in a geometric manner, iso-surface scalar fields and allow the generation of path lines. One might view the development of such visualization efforts similar to general-purpose math libraries, however, even with general-purpose math library, we still have solvers developed and customized for specific problems. These solvers take into consideration knowledge of the domain experts. Why shouldn't the analysis tools incorporate similar knowledge in order to reduce the data to the features of primary interest?

Model centric

This paper proposes that analyzing and understanding the results of large-scale simulations should be from the model's perspective rather than from a data-centric point of view. It is, after all, the mathematics and physics that underlie the simulation from which we seek to understand and gain knowledge. The visualization, for these problems, is a means to assimilate information from the model. However, large-scale simulations have too often been associated with relatively scarce compute resources leading us to the following problems/misconceptions:

1. The analysis of large-scale simulation data is a separate process from its generation.

2. The raw output data of the simulation is precious and every bit should be saved.

3. In order to view the simulation results, big (expensive) visualization systems are required.

The purpose of this paper is to dispel these misconceptions and provide a road map (by adopting the model's perspective) in which the knowledge embedded within large-scale simulations can be extracted. These techniques can be thought of as physics based data-mining.

Tradition Approach Vs Simulation-Based Approach

There has been some effort in coupling the post-processing visualization phase with the solver phase using extraction techniques to permit interactive visualization on distributed parallel computing systems.

Much of this effort is embodied in the **pV3** interactive visualization toolkit.[1–4] However, in order to find important flow features, the domain expert must interactively search his data. Particularly for large-scale 3D simulations, which are often run in a batch-processing mode, there are drawbacks to both the post-processing visualization mode and the interactive concurrent visualization mode.

The post-processing visualization mode suffers from:

1. Saving the entire volume of data per time step results in a large amount of data storage most of which is not of any value.

2. Predetermined time steps of the transient simulation are saved. Important feature evolution may be missed as a consequence of insufficient number of time steps saved. Also, some general visualization that requires a time history may produce inaccurate results (i.e. particle integration).

3. Inability to probe and interrogate the data and no opportunity for redirecting the simulation, i.e. steering.

Current interactive visualization has some drawbacks for large-scale simulations:

1. Interactive visualization techniques produce output based on local sample points in the grid or solution data. Important features may be missed if the user does not exhaustively search the data set.

2. The interactive techniques usually reveal the flow behavior in the neighborhood of a flow feature rather than displaying the feature itself.

3. Interactive exploration of very large data sets is laborious and consumes too much time to be practical.

4. After generating only a small number of visualization objects the display becomes cluttered and makes visual interpretation difficult.

There is now some activity in flow visualization research focused on developing feature extraction algorithms that automate the data analysis and permit the extraction of important features. However, one obstacle for the development of such automatic feature extraction algorithms has been precise mathematical definitions for flow structures of interest to the domain expert. Deriving appropriate mathematical formulations germane to the domain expert often requires a deep mathematical comprehension of the simulation. However, once such a formulation is determined, direct feature extraction has the following advantages over the traditional exploratory post-processing and interactive visualization tools:

1. Deterministic Algorithms
   If there are no "parameters" that the user may need to adjust, then no intervention is required.

2. Fully Automated
   The analysis can be done off-line in a batch computation. It can be used directly by a solver to adapt the mesh for a better resolution.

3. Local Analysis
   These schemes, where possible, perform only local operations. Therefore, the computations for each cell are independent of any other cell and may be performed in parallel. This is clearly advantageous in distributed memory compute arenas.

4. Data Reduction
   The output geometry is several orders of magnitude smaller than the input data set. This is an important characteristic for the size of a resultant output. High fidelity spatial and temporal results of the feature extraction can be stored on disk for concurrent extraction coupled with the solver. The side effect is that playback is rapid (the extraction process is done and the data has been distilled to the salient information). Better than a movie, the viewpoint can be changed during the playback.

5. Quantitative Information
   Precise locations for the flow features are extracted. Classification and measures of strength can be reported.

The results of the feature extraction can be viewed in a 3-D interactive visualization environment and can be used in conjunction with interactive visualization tools. Some of these tools have been developed in the **pV3** toolkit and have been used in many problems, which include groundwater remediation, electromagnetic radiation in cellular phones and mantle dynamics.[5]

The approach outlined in this paper requires these major technologies:

1. Extracts.
   A method for dealing with the data from within a distributed setting.

2. Co-processing.
   Concurrent filtering of the simulation data so it's size can be reduced by many orders-of-magnitude.

3. Batch co-processing.
   Collecting the filtered data when the simulation runs.

4. Feature extraction.
   Answering specific questions of the data during the simulation.

Extracts

Most traditional visualization algorithms are embarrassing parallel. This is because at some level most operations are performed on an element and these cells are (or can be easily) distributed throughout the compute arena. Little work needs to be done to take the current tools and distribute the compute portion to where the data resides. The REAL problem is how to minimally describe the result of the tool and move it to where it is needed. A fine discussion of this concept, 'extracts', can be found in a treatise by Globus.[6] 'Extracts' are *NOT* graphics but are objects that contain enough information to generate the graphic. A graphic object is inflexible. It contains vertex information, such as coordinate position and color, and polygonal data (and optionally surface normals) so the graphic can be included in the final scene. 'Extracts' contain the coordinate data but color is not assigned. Scalar and even vector data can be assigned to each vertex. The associated cell information can be used to generate the graphic and also used for further probing.

'Extracts' allow the type of freedom required for a general distributed visualization system. For example, first an iso-surface is generated. Now, to render it with the value of another scalar field, only the color mapping need be changed. This does not require re-computing the iso-surface, but while the graphic is generated (during immediate-mode rendering), the color associated with each vertex is changed. The only new data required is the scalar field values at each vertex of the object.

Clearly, producing 'extracts' where the volume of data exists and only passing the results at a reduced dimensionality does not add much of a message-passing burden. Only the data required to draw the result of the tool and/or probe is moved over the network. Reducing the data by at least one dimension is the key that allows a distributed system to function and scale.

This also frees up the compute cluster from all graphics. Rotations, panning, scrolling and zooming (any change of the viewing transformation matrix) are handled by the graphics workstation locally (with no new data required over the network). This is also the case for a change in scalar field mapping and vector (or tuft) length.

Co-Processing

Concurrent solving with interactive visualization can relieve the compute arena of the disk space, speed and sampling issues. It does require a complete rethinking of the architecture of the visualization system and poses the following issues:

- Coupling to the solver
  The solver must communicate with the visualization system. This can be accomplished by one of three methods:
  - Disk files: In this approach the solver task communicates with the visualization task by data written to disk. This method is rejected because of the disk timing and I/O arguments.
  - Shared memory: In this approach the solver and visualization system communicate via shared memory. This has the advantage that the solver task and visualization task are separate and the communication is fast. The disadvantages are that the solver must be written with the shared memory interface, the data that is exposed must be done in a way that the visualization task knows where and how to get individual data elements. Also, some method is required to mark the data invalid as it is being updated.
  - Application Programming Interface (API): This method couples the visualization task (or some portion of the visualization system) with the solver. This coupling is done at the programming level. The advantages to this approach are that all the data is shared (there is only one task), no special system level requirements are needed, and it can be used with solvers written in different languages. The challenge is to develop a software architecture that is sufficiently general, non-invasive and allows the visualization system and solver to function independently.
- Additional resources
  Whichever approach is selected, an additional burden is placed on the computer resources. Now both the solver and at least a portion of the visualization system are active on the computer. The visualization subsystem should not place a large encumbrance on either the memory requirements or need large numbers of CPU cycles. Also, it should not transform a compute bound problem into an I/O bound problem.
- Robustness
  The visualization system (or a part of it) is running concurrently with the solver. Either it is part of the same task, or has access to crucial data. The visualization must be robust and not interfere with the solvers functioning. A problem with the visualization portion must not crash the solver.
- Speed
  It is important for interactivity that the resources required by a co-processing visualization system be a minimum. The cost of CPU cycles and additional memory are the barometer for the frame rate. To accomplish fast frame rates, it becomes a necessity to use the classification of transient

problems. If the visualization system is properly designed, this classification can be used to determine when the results of a tool should be saved and when recalculation is necessary.

An example of a co-processing distributed visualization system is **pV3**. The design of **pV3** allows the solver to run on other equipment than the graphics workstation. In fact, the development of this tool-kit was in the cluster environment with **PVM** used for process control and as the message passing subsystem.

pV3 segregates the visualization system into at least two parts. For simple set-ups, the first part is the task that actually displays the visualization results onto the graphics workstation; the server (using X-windows terminology). The second portion is a library of routines that allows the solver (solver portions or solvers) to communicate with the visualization server by providing access to the data at critical times during the simulation. This client-side library separates all visualization-based calculations from the solver, so that the solver programmer only needs to feed **pV3** the current simulation data.

pV3 has been designed to minimize network traffic. The client-side library extracts lower dimensional data required by the requested visualization tool from the volume of data in-place. This distilled data, the extracts (as described above), are transferred to the graphics workstation. To further reduce the communication burden posed by the visualization session only the extracted data that has changed from the last iteration is sent over the network.

An added benefit of this architecture is that most of the visualization compute load is run in parallel for a cluster or Massively Parallel Processor (MPP) environment. This is because most visualization tools are based (or can be cast) on operations performed on individual cells within a partition. This means that these tools are embarrassingly parallel. If the partitions are balanced on the basis of cells, then the result of the visualization computation is also executed in parallel with some load balancing. Therefore much better performance for the visualization is achieved.

To maximize **pV3**'s effectiveness, client-side libraries exist for most major PC and workstation vendors, MPPs and super-computer manufacturers. **pV3** has been designed so that the process of coupling to the solver is simple as possible and noninvasive. The solver's control structure is maintained with the visualization subservient. In the distributed setting, **pV3** client(s) perform their appropriate calculations for the problem, then as the solution is updated, supplied call-back routines are executed by the visualization library.

Coupling to the **pV3** client-side library is performed by adding two **pV3** calls to the solver application or simulation code. The first call informs the visualization system about the volume discretization, the kind of transient problem, the visualization field variables (i.e., the number, type, name and limits for each scalar, vector and threshold fields) and additional geometric cuts defined for this case.

This second access point (a call to PV_UPDATE) informs the visualization system of two things; (1) the simulation time for time-accurate solvers and (2) the solution has been updated and synchronized. This is the mechanism that allows **pV3** to operate. Without the server running, every time the solution is updated and this call is executed, a check is made for any members in the server group. If none are found, this call just returns resulting in essentially a no-operation. When the visualization task starts, it enrolls in the server group, and waits for initialization messages from the clients. The next time the client goes through an iteration, and calls the **pV3** update routine, an initialization message is sent to the server. The server processes all initialization messages, figures out the number of clients and the visualization session begins. Each subsequent time the client goes through the loop and makes this **pV3** call, visualization state messages and tool requests are gathered. Then the appropriate data is calculated, collected and sent to the graphics workstation.

When the user is finished with the visualization, the server sends a termination message, leaves the server group, and exits. The clients receive the message and clean up any memory allocations used for the visualization. The **pV3** client update call reverts to looking for members in the server group.

Batch-style co-processing

There is a problem using **pV3** in production environments or for batch execution. The difficulty is that the user may not be around to start the interactive **pV3** server and view the results. Also, even if someone is looking at the data, when the image on the screen is updated, the old information is not saved for later retrieval.

To resolve this issue another **pV3** module can be accessed – the 'batch' server. In this case, the client side remains totally unchanged. The coupled solver need not know if the results are currently being viewed or are to be viewed at some later time.

When a batch job starts, the batch **pV3** server is also started. Data is read on what tools and probes are to be activated and their locations. The results (the tool extracts) are collected and written to disk for later playback. This is different from the normal post-processing in that the entire volume of data is not written to disk.

The end result is something that is not interactive in the placement of tools, but can be thought of as analogous to an experiment. Some knowledge of the expected areas of interest must be used in the placement of probes to extract data of interest. If important information is missed (or only found after viewing these results) the numerical experiment will have to re-run adding (or changing the location) of the probes.

The 'post-processing' **pV3** viewer module becomes highly interactive in dealing with time. This is because the extraction is complete and the amount of data has been reduced by orders-of-magnitude. Overlapping the disk I/O with display rendering further enhances the viewer's performance. This is accomplished via multi-threading. The viewer is a self-contained module (a complete executable with no other dependence). Therefore the simulation can be reviewed away from the hardware used in data generation. All that is required is the resultant disk file, the post-processing viewer and the proper graphics workstation.

This mode of operation has these additional advantages:

- The data is collected.
  The process of patching together field data from a simulation that has gone through domain decomposition can be time consuming. This is not required for this type of visualization. The data is already collected and deposited in a single file.

- Visualization is done at the end of the simulation.
  No scripts are needed. The collection of visualization data has been stored. All that is required, at this point, is to view and attempt to understand the results from the simulation.

- Need for videos is reduced.
  The movie is already made, but using this paradigm the viewpoint and plotting attributes for the collected extracts can be interactively changed while playing back the results.

The batch server and post-processing viewer can be thought of as the interactive **pV3** server split into two portions. These parts communicate through the Generic Engineering Data Structure (GEDS) data file. This file format is documented at the byte level and described in the **pV3** distribution.

Feature Extraction

In the past, feature extraction and identification were viewed as interesting concepts. But they were not required for understanding the underlying physics of steady-state problems. This is because the results of the traditional tools like geometric cuts, iso-surfaces and streamlines were more interactive. These tools worked and properly conveyed the collected information at the expense of much user interaction and inference from the resultant imagery. For unsteady simulations, the investigator does not have the luxury of spending much time scanning only one "snap-shot" in the simulation. Automated assistance is required in pointing out areas of potential interest. It is also crucial to develop methods that abstract the feature and display it in an interactive and visually concise manner.

Limited work is underway to develop algorithms and the code that can find fluid features from CFD results. It should be noted that present feature detection techniques reduce the data by two to four orders of magnitude. Therefore, multi-gigabyte data sets can be reduced to tens of megabytes of 3-D geometry. The raw geometric output from these techniques can be further reduced using well known line and surface simplification techniques from computer graphics and Computer Aided Design fields. The expectation is that this will provide a five to six orders of magnitude reduction, thereby reducing terabytes to megabytes, and make it possible to interactively view the results on a modest workstation.

When feature techniques are linked with an extract based (batch) co-processing visualization subsystem then the simulation based data-mining system is complete. Answers to specific questions can be distilled, in place, and transmitted from the solver with little impact on the overall computer resources.

It should be noted that the results from these feature techniques are geometry and not imagery. Therefore there are no difficulties with multi-scale physics. One can continue to zoom in to the feature and it will be resolved to the resolution of the underlying simulation data.

Some of the current feature extractors for fluid dynamic simulations are:

- Shocks
  The display of shocks is simple; a shock is a surface in 3-space. As the solution progresses the investigator can view the changing shape of the shock surfaces. A recent overview of transient shock finding[7] concludes that using the fluid mechanics definition of a shock is preferable to the 3-D analogue of edge detection in image processing.

  For a steady-state shock an iso-surface of the normal Mach number can be used:

  $$M_n = \frac{\vec{u}}{a} \cdot \frac{\nabla p}{|\nabla p|} = \vec{M} \cdot \frac{\nabla p}{|\nabla p|} = 1.0$$

  Shock's frame of reference:

  $$\frac{1}{|\nabla p|}\frac{1}{a}\frac{\mathbf{D}p}{\mathbf{D}t} = \frac{1}{|\nabla p|}\frac{1}{a}\frac{dp}{dt} + \vec{M} \cdot \frac{\nabla p}{|\nabla p|}$$

  For isentropic flows:

  $$dp = a^2 d\rho$$

  $$\frac{dp}{dt} = -a^2 \nabla \cdot (\rho\vec{u})$$

$$\underbrace{-a\frac{1}{|\nabla p|}\nabla\cdot(\rho\vec{u})}_{transient-correction} + \underbrace{\vec{M}\cdot\frac{\nabla p}{|\nabla p|}}_{steady-state} = 1.0$$

The generalized correction term needed for transient problems:

$$p = (\gamma-1)[\rho E - \frac{1}{\rho}(\rho\vec{u})\cdot(\rho\vec{u})]$$

Take the time derivative:

$$\begin{aligned}\frac{dp}{dt} &= (\gamma-1)[\frac{d}{dt}(\rho E) - \\ &\quad \vec{u}\cdot\frac{d}{dt}(\rho\vec{u}) + \frac{1}{2}\vec{u}^2\frac{d\rho}{dt}]\end{aligned}$$

$$\begin{aligned}\frac{dp}{dt} &= (\gamma-1)[-\bigtriangledown\cdot(\rho\vec{u}H) + \\ &\quad \vec{u}\cdot(\bigtriangledown p + \bigtriangledown\rho\vec{u}\vec{u}) - \frac{1}{2}\vec{u}^2\bigtriangledown\cdot(\rho\vec{u})]\end{aligned}$$

The derived quantity is iso-surfaced at a value of 1.0. The resultant surface bounds the region that contains the shock.

- Vortex Cores
  Vortices can be automatically detected by using the velocity gradient tensor ($\underline{V}$) throughout the mesh looking for situations of swirling flow. All mesh elements are broken into tetrahedra (if not already this type of element). The unique $\underline{V}$ is constructed and then classified. If swirling, the direction orthogonal to the spiral plane (the eigenvector associated with λ_r) is used as the axis of swirl. This direction is subtracted from the nodal velocities. These *reduced velocities* are used to see if any faces display a zero. If so, that location on the face is marked. With two (or more) marks on the tetrahedron's face, it is determined that the core centerline has pierced the cell. These lines are collected and drawn to display the core segments.

  This particular algorithm is fully described in Sujudi and Haimes[8] and contains no constants and requires no user intervention. The core finder has been applied to a fairly complete suite of applications, see Kenwright and Haimes.[9]

  This technique, although satisfying, does not work under all circumstances. The problems are:

  - Not producing contiguous lines.
    The method, by its nature, does not produce a contiguous line for the vortex core. This is due to two reasons; (1) for element types that are not tetrahedra the interpolant that describes point location within the cell is not linear. This means that if the core passes through these elements the line can display curvature. By subdividing pyramids, prisms, hexahedra and higher-order elements into tetrahedra for this operation produces a piecewise linear approximation of that curve. (2) there is no guarantee that the line segments will meet up at shared faces between tetrahedra. This is because the eigenvector associated with the real eigenvalue will not be exactly the same in both neighbors, so when this vector is subtracted from the vector values at the shared nodes each tetrahedra sees a differing *reduced velocity* field for the face.
  - Locating flow features that are not vortices.
    This method finds patterns of swirling flow (of which a vortex core is the prime example). There are other situations where swirling flow is detected, specifically in the formation of boundary layers. Most implementations of this technique do not process cells that touch solid boundaries to avoid producing line segments in these regions. But this does not always solve the problem. In some cases (where the boundary layer is large in comparison to the mesh spacing) this boundary layer generation is still found.
  - Sensitivity to other non-local vector features.
    Critical point theory gives one classification for the flow based on the local flow quantities. 3-D points can display a limited number of flow topologies including swirling flow, expansion and compression (with either acceleration or deceleration). The flow outside this local view may be more complex and have aspects of all of these components. The local classification will depend on the strongest type. Also if there are two (strong) axes of swirl, the scheme will indicate a rotation that is a combination of these rotation vectors based on the relative strength of each. This has been reported by Roth and Peikert[10] where the overall vortex core strength was not much greater that the global curvature of the flow. The result was that the reported core location was displaced from the actual vortex.

The first point can be addressed by re-casting the algorithm to be face-based instead of cell-based. Enforcing the cell piercing to match at cell faces ensures that the line segments generated will produce a contiguous core. This can be done via the following modification to the algorithm:

1. Compute the velocity gradient tensor at each node.
2. Perform the eigen-analysis on the node tensor.
   The tensor can be overwritten with the critical point classification and the swirl axis vector for rotating flow.
3. Average the swirl axis vectors for the nodes that support the tetrahedral face.
   This should only be done if all nodes on the face indicate swirling flow. Some care needs to be taken to ensure that the sense of the vectors is the same. Determine if the swirling axis cuts through the face, and if so, mark the location on the face.

For tetrahedral meshes, the reduction of compute load is by a factor of 5 to 6 over the original method (there are roughly 5.5 tetrahedra per node in 'good' unstructured grids). For structured blocks, where the number of nodes is about equal to the number of hexahedra, the eigen-analyses count is on the order of one per cell.

This new algorithm is still linear and will produce incorrect results when the flow is under 2 or more (relatively equal) rotating influences (the third problem listed above). Roth and Peikert[11] presented a paper that, on first viewing, appears to solve this problem. The authors suggest that by looking at higher-order derivatives of the velocity field one can capture curvature. They first recast the technique described above then apply the higher-order correction:

– Parallel alignment
  An intersection point on a face is where the *reduced velocity* is zero. Therefore the velocity vector is parallel to the real eigenvector of $\underline{V}$. $\vec{u}$ is an eigenvector of $\underline{V}$ and therefore a solution of
  $$\underline{V}\vec{u} = \lambda\vec{u}. \tag{1}$$
  This suggests that looking for parallel alignment of $\underline{V}\vec{u}$ with $\vec{u}$ is the same as the vortex algorithm described above.

– Second Order
  Computing the velocity second derivatives ($\mathbf{T}$) generates a $3x3x3$ tensor. Adjusting Equation (1) for the second derivative produces:
  $$\underline{V}\,\underline{V}\vec{u} + \mathbf{T}\vec{u}\vec{u} = \kappa\vec{u}. \tag{2}$$

– Alignment with Second Order Correction
  Equation (2) is used to find where in the flow field $\underline{V}\,\underline{V}\vec{u} + \mathbf{T}\vec{u}\vec{u}$ is parallel to $\vec{u}$. These locations are marked and collected like the eigen-analysis technique.

In practice, this does not seem to work well. The velocity field generated by most CFD solvers is not smooth. As a result this technique produces both many false positives and also misses intersections due to the local saw-tooth variations. The other problem is the storage requirements. For large data sets, requiring 27 floating point words/node of the rank 3 tensor and 9 floating point words/node for the rank 2 tensor becomes prohibitive.

The issue of alignment coupled with the linear limitations of the classification and results of the eigen-analysis hint at an alternative technique. This approach uses $eig(\underline{V}-\underline{V}^T)$ instead of $eig(\underline{V})$. The result is invariably one zero and a complex conjugate pair (λ_c) of eigenvalues – always the indication of swirling flow. The eigenvector associated with the zero eigenvalue can be used with the face- piercing algorithm to produce core segments. This technique has the effect of removing the bulk and shear components of stress from the analysis.

The effect of this algorithm is, at times, longer and more contiguous cores. For example, tracking a horseshoe vortex in a turbine cascade, the core line would disappear as the flow accelerated around the crown of the suction surface. This was due to a change in classification from swirl to non-swirling flow. With the irrotational components removed the core can continue to be extracted. The influence of the irrotational terms also explains why the core segments would break-up when the core strength would diminish. Under these conditions the stress components end up corrupting the direction of the eigenvector used for the swirl axis.

- Regions of Recirculation
  Recirculation is a difficult feature to locate, but a simple one to visualize. A surface exists that separates the flow (in steady state) so that no streamlines seeded from one side of this surface penetrate the other side. This definition does not work for transient problems. Like a series of instantaneous streamlines can be misleading in unsteady flow regimes, using techniques based on streamlines will not represent the regions of stagnant fluid. The concept that appears correct is Residence Time. This is the Eulerian view of unsteady particle tracing (a Lagrangian operation). A simple PDE can be solved on the same mesh along with the flow solver. (NOTE: This is only possible when performing co-processing do to the same time limit constraints.) An iso-surface can

be generated through the result so that regions of *old* fluid can be separated from *newer* fluid elements.

The residence-time of a volume of inviscid fluid is defined by

$$\frac{\mathbf{D}\mathcal{T}}{\mathbf{Dt}} = 1 \tag{3}$$

where $\mathcal{T}$ denotes residence-time. This is similar to conservation of mass, but with a source term. Since $\frac{\mathbf{D}}{\mathbf{Dt}} \equiv \frac{\partial}{\partial t} + \vec{u} \cdot \nabla$, then Equation (3) becomes

$$\frac{\partial \mathcal{T}}{\partial t} + \vec{u} \cdot \nabla \mathcal{T} = 1 \tag{4}$$

where $\vec{u}$ is the velocity vector.

Since the time when the residence-time calculation starts is defined as time zero, then initial the condition is

$$\mathcal{T}(x, y, z) = 0. \tag{5}$$

At in-flow boundaries, new fluid is entering. By definition, this fluid has zero residence-time. Therefore, the boundary condition is

$$\mathcal{T}(x, y, z) = 0 \qquad at\ in-flow. \tag{6}$$

To obtain the conservative form of Equation (4) for incompressible flows, this can be rewritten as

$$\frac{\partial \mathcal{T}}{\partial t} + \vec{u} \cdot \nabla \mathcal{T} + \mathcal{T} \nabla \cdot \vec{u} = 1 + \mathcal{T} \nabla \cdot \vec{u}$$

or

$$\frac{\partial \mathcal{T}}{\partial t} + \nabla \cdot (\mathcal{T} \vec{u}) = 1 + \mathcal{T} \nabla \cdot \vec{u}.$$

And since $\nabla \cdot \vec{u} = 0$ for incompressible flows, then

$$\frac{\partial \mathcal{T}}{\partial t} + \nabla \cdot (\mathcal{T} \vec{u}) = 1. \tag{7}$$

Equation (7) reflects the residence-time in an analogous manner to streaklines. Here the result is only a function of the velocity field. Iso-surfacing $\mathcal{T}$ with a value that is greater than the characteristic time of the domain marks the region of recirculation. A complete description of this algorithm and how it can be solved can be seen in.[12]

It should be noted that these three examples display a progression. The shock function can be calculated from local information and could be computed from the field calculator supplied in many visualization systems (at least for the steady-state marker). The vortex core finder requires numerics and operations not found in these calculators. But, it still does not require a temporal history for transient simulations. Finally, the recirculation finder begins to look like the simulation itself. The solver/visualization/data analysis loop is being closed, and that is the point. We are solving a specific problem; if the traditional visualization tools do not answer the investigators questions then recast the simulation so that can be accomplished.

The residence-time formulation is the archetypical example of a series of possible Eulerian markers. By including a different right-hand side, Equation (3) can be recast to:

$$\frac{\mathbf{D}\mathcal{B}}{\mathbf{Dt}} = f(F_v)$$

This equation can track the difference between the Euler and the Navier-Stokes Equations where F_v reflects the viscous stresses. It could be iso-surfaced at some value slightly above zero to show the extent of the boundary layers and wakes found in the simulation. There are two formulations: (1) involving only the terms in the momentum equation and therefore $\mathcal{B}$ would track a momentum deficit and (2) the viscous terms found in the energy equation where $\mathcal{B}$ would be reflective of the thermal boundary layer.

Another marker is:

$$\frac{\mathbf{D}\mathcal{M}}{\mathbf{Dt}} = \vec{u}$$

Where $\mathcal{M}$ is a measure of the mobility of the fluid past a location. The result is the distance that that the fluid has moved from the beginning of the simulation. This can be used to track porosity and/or containment. For example, if one is using a simulation to ask the question: is the Yucca Mountain Repository suitable for the long-term storage of nuclear wastes? Then, $\mathcal{M}$ would be an appropriate marker. By looking at the values of $\mathcal{M}$ at the bounds of the repository one can easily see if there is any migration of fluid. And, if so, it can be tracked back to the source.

Conclusions

By following what is prescribed in this paper one can effectively and efficiently work with large-scale scientific simulations. This 'roadmap' allows one to gain insight, to formulate the desired conclusions, and ultimately produce the understanding and knowledge from the simulations. Visualization transitions from peaking and probing the results to producing imagery that directly answers the investigator's questions and consequently makes the simulation more useful.

Many research and commercial visualization packages contain field calculators. It was assumed (or hoped) that this tool could be used for feature extraction. In this way, if the user understands the field variables, then physics can be brought back into the analysis. Unfortunately, from the discussion in the Feature Extraction Section, it is clear that calculators can not perform the functions required. Only the steady-state portion of the shock finder can be computed by most calculators and there is a disturbing side effect. The memory usage is large. Mach number, pressure and velocity are required (5 words/node) and 3 temporaries are needed. These are Mach vector, pressure gradient (a vector) and pressure gradient magnitude consuming another 7 words/node. If the problem is large, then this additional memory requirement is problematic.

As has already been pointed out, the solvers used in large-scale simulations are often application dependent. A domain expert (often working with a computational scientist) develops application specific code. Numerical analysis 'tricks' as well as distributing the task to an MPP may be required to make the simulation tractable. Why, then, should we expect to use a generic suite of visualization tools to understand the results from this specific simulation? Wouldn't application specific visualization permit greater and more meaningful insight? In part, the issue is that the visualization tool developer and the domain expert are not as closely connected as the solver developer and the domain expert. Too much emphasis is placed on producing an image from the data of the simulation instead of gaining knowledge and then being able to discard much of the data (that can be and often is for some reason regenerated anyway).

This leads to the proposal that by 'throwing away' the following precepts, we can make a seeming impossible problem understandable:

- The analysis of the data is a separate process from its generation
- The raw output data is precious
- We need the 'biggest' visualization systems to view the results
- We must treat our 'big iron' like batch systems

What this paper simply suggests is that the process of analyzing and understanding the results of these large-scale simulations must not be thought of from a data-centric viewpoint, but from the model's perspective. This challenges the visualization community (and funding sources) to redirect personnel resources from disk and data management tasks to domain specific feature research. This is an important shift for the future. If the past can be used as a predictor, the size of these simulations will continue to grow and outpace our ability to handle the resultant data.

Finally, this question needs to be asked: How does volume rendering fit into this approach. The answer is simple – nowhere. Volume rendering is an example of where the visualization has limited the information available from the simulation. While volume rendering provides an effective means for generating the visual, more inference is required to understand the resultant imagery. In addition, in a volume rendered image, one does not easily have the ability to point-to or probe the picture to get the sought after feature of interest. This is a consequence of the lack of geometric information. Zooming in on areas that may have interesting features becomes much more difficult and information provided is harder to comprehend. Volume rendering provides a visual that is further removed from the physics, exactly the opposite effect desired for understanding and gaining knowledge from the simulation. In the case of multi-scale or multi-physics problems, volume rendering minimizes our understanding by smearing the multi-scale or multi-physics effects making the interactions, the important information, harder to discern.

References

[1] R. Haimes. **pV3**: A Distributed System for Large-Scale Unsteady Visualization. AIAA Paper 94-0321, Jan. 1994.

[2] R. Haimes. Unsteady Visualization of Grand Challenge Size CFD Problems: Traditional Post-Processing vs. Co-Processing. Proceedings of ICASE/LeRC Symposium on Visualizing Time-Varying Data, Sept. 1995.

[3] R. Haimes. A Modular Approach to Visualization for Parallel CFD Applications. Proceedings of Parallel CFD '96, May 1996.

[4] R. Haimes and D. Edwards. Visualization in a Parallel Processing Evnironment (invited). AIAA Paper 97-0348, Jan. 1997.

[5] K.E. Jordan, D.A. Yuen, D.M. Reuteler, S. Zhang and R. Haimes. Parallel Interactive Visualization of 3D Mantle Covection. IEEE Computaional Science and Engineering, Vol. 3, No. 4, Winter 1996.

[6] A. Globus. A Software Model for Visualization of Time Dependent 3-D Computational Fluid Dynamics Results. NASA Report RNR-92-031, November 1992.

[7] D. Lovely and R. Haimes. Shock Detection From Computational Fluid Dynamics Results. AIAA Paper 99-3285, Jun. 1999.

[8] D. Sujudi and R. Haimes. Identification of Swirling Flow in 3-D Vector Fields. AIAA Paper 95-1715, Jun. 1995.

[9] D. Kenwright and R. Haimes. Vortex Indetification – Applications in Aerodynamics. IEEE Computer Society, Visualization '97, Oct. 1997. Awarded 'Best Case-Study'.

[10] M. Roth and R. Peikert. Flow Visualization for Turbomachinery Design. Proceedings of IEEE Visualization '96, 1996.

[11] M. Roth and R. Peikert. A Higher-Order Method For Finding Vortex Core Lines. Proceedings of IEEE Visualization '98, 1998.

[12] R. Haimes. Using Residence Time for the Extraction of Recirculation Regions. AIAA Paper 99-3291, Jun. 1999.