Visualization Viewpoints

Editor:
Theresa-Marie Rhyne

The Tensions of In Situ Visualization

Kenneth Moreland
Sandia National Laboratories

In situ visualization is the coupling of visualization software with a simulation or other data producer to process the data "in memory" before they are offloaded to a storage system. This is in contrast to the more traditional workflow where the writing of simulation data from memory to disk completes before the visualization starts and reads the data from disk back into memory.

In situ visualization is a critical technology for scientific discovery with exascale computing.[1,2] There are numerous advantages to in situ visualization, but the most pressing feature is the ability to remove the bottleneck of disk-based I/O. As Figure 1 illustrates, the bandwidth of a modern supercomputer's storage system is a tiny fraction of the compute bandwidth, and this discrepancy is expected to grow. Although the amount of data that can be stored is limited by the storage system capacity, its bandwidth also limits how much data can feasibly be written (and read later during interactive visualization).

Consequently, in situ visualization is an active R&D topic for high-performance visualization, as the images in Figure 2 show. Although in situ visualization provides superior analysis, it has implementation tradeoffs resulting from conflicts with some traditional expected requirements. Numerous conflicting requirements create tensions that lead to difficult implementation tradeoffs. This article takes a look at the most prevailing tensions of in situ visualization.

Batch Versus Interactive

One of the biggest limitations of in situ visualization is that the data are transient. Data are available during a limited window in time, and after that, the data are lost. This is in contrast to traditional visualization where all the available data can be loaded from disk storage at will.

This makes interaction and exploratory visualization challenging in an in situ environment. Large-scale simulations can run for days, and it is likely not feasible for someone to be interactively visualizing that entire time. Furthermore, even if someone is actively analyzing the data, there is no assurance that that person will find all salient information during the time the data are available.

Consequently, much in situ visualization focuses on automated batch processing. In this case, the visualization is predetermined; it is automatically executed whenever appropriate, and the results are stored for later analysis. Although batch processing allows visualization to occur at a fine temporal fidelity, it is restricted to analysis specified a priori. Because neither batch nor interactive modes satisfy all use cases, both are commonly used with in situ visualization, and production packages like ParaView Catalyst and VisIt libsim support using both simultaneously.

Because neither batch nor interactive in situ visualization is fully satisfactory, a fruitful area of research is using in situ processing to make the most of the data that are stored on the file system for post hoc analysis. One approach is to find a more compact representation of the data. Extracting regions of interest, subsetting, contouring, and compressing are all straightforward examples of compacting the representation. Feature extraction techniques, which find abstract representations of data that require less memory to characterize, are also effective.[3] An alternate (and perhaps complementary) technique to transforming the data is to find important or representative samples in space and time to maximize the uniqueness of the data that does get stored.[4]

Another approach to bridging the gap between batch in situ processing and interactive post hoc analysis is the generation of visualization artifacts that are interactive content rather than static imagery. For example, explorable images[5,6] use special rendering techniques that allow the images to be later manipulated as if the original data were still

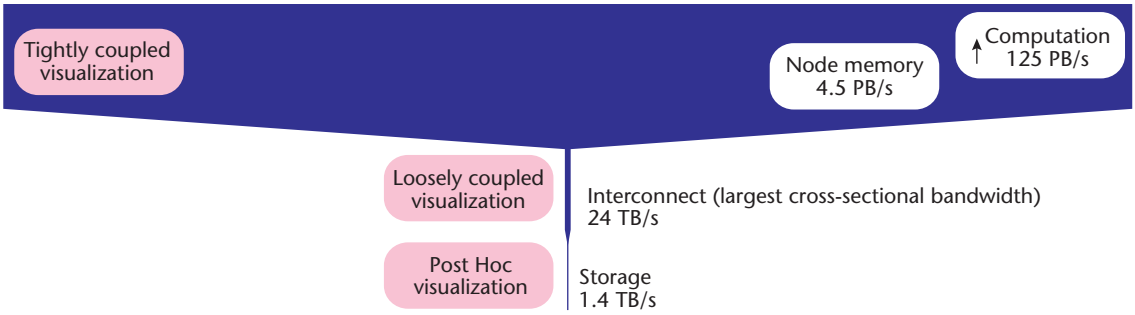

Figure 1. The relative theoretical aggregate bandwidth of different system components on the Titan supercomputer at the Oak Ridge Leadership Computing Facility. The widths of the blue boxes are proportional to the bandwidth of the identified components. The bandwidth of the storage system is dramatically slower than the interconnect, which is dramatically slower than the bandwidth of the on-board memory, which is dramatically slower than the cache and registers.

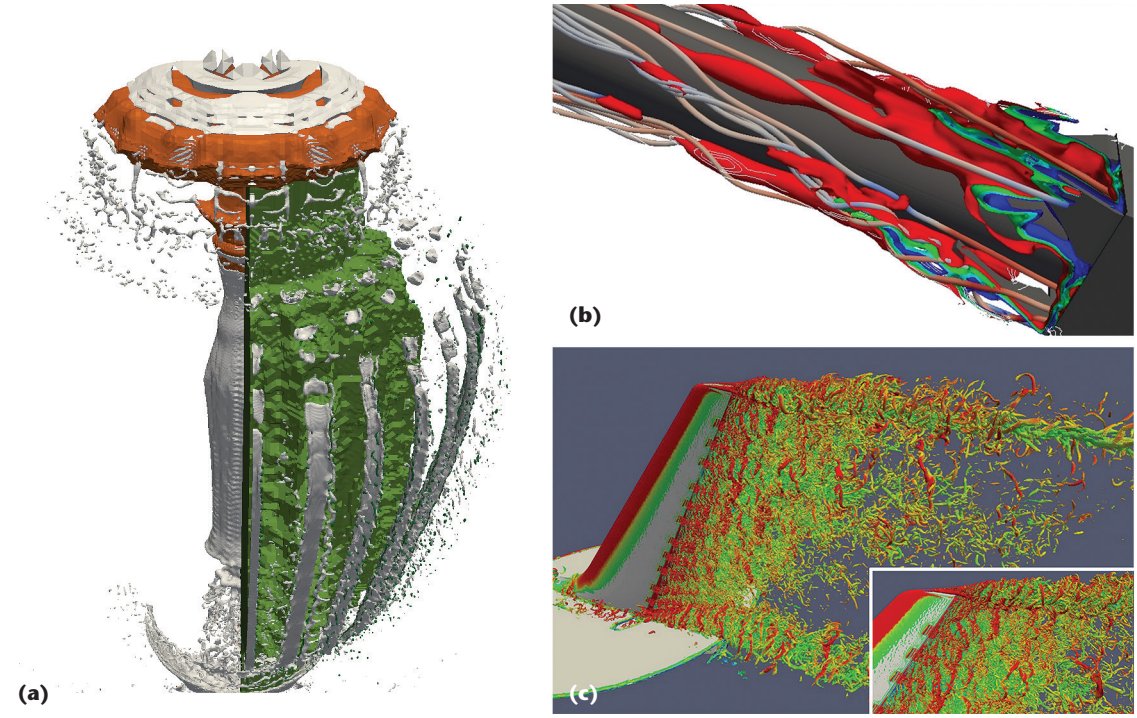

Figure 2. Examples of in situ visualization. (a) A CTH shock physics simulation (courtesy of Nathan Fabian, Sandia National Laboratories), (b) a Hydra-TH thermal hydraulics simulation (courtesy of Mark A. Christon, Los Alamos National Laboratory), and (c) a PHASTA CFD simulation run and visualized on 256,000 MPI processes (courtesy of Michel Rasquin and Kenneth Jansen, University of Colorado Boulder).

available, as Figure 3 shows. Another project, titled Cinema, collects visualization results with varying parameters and places them in a database that can be interactively explored like that shown in Figure 4.[7] The approach recognizes that it is not always possible to specify a priori what may be of interest in a visualization so the possible visualization parameters space is sampled to capture unforeseen areas of interest and insight.

Tight Coupling Versus Loose Coupling

With the exception of small, custom visualization systems, different software teams develop the simulation code and the visualization code. As such the coupling of these two software systems can lead to cultural clashes as well as technical challenges. This leads to tension in how tightly simulation and visualization should be integrated. A tightly coupled visualization will be embedded in the simulation process sharing all the same resources. A looser coupling will have the simulation and visualization as two separate communicating processes. The simulation and visualization may share the same cores, use separate cores on the same computer nodes, or communicate across separate nodes. Loosely coupled in situ visualization often has middleware managing the connection.[8] (A purist might argue that only a tight coupling meets the "in place data"

meaning of in situ, but for the purposes of this article, we consider any visualization before the data are written to disk storage as in situ.)

Tightly coupled in situ visualization is generally the most efficient. Shared resources give the fastest access to data (as Figure 1 demonstrates) with the minimum amount of replication, and reduced communication can even lead to power savings.[9] Also, tightly coupled visualization has the potential to access simulation metadata that is difficult to share in loose coupling but can greatly improve performance.[10]

However, tight coupling has its costs. A failure in one component is almost certain to bring down other tightly coupled components, so tight coupling can make development teams more wary. The visualization's use of memory and the network can potentially negatively affect the simulation's performance. Also, the visualization algorithm may perform better at a smaller scale than the simulation, and separating the two can sometimes lead to better overall performance.[10,11]

The upshot is that although visualization tools like Catalyst and libsim are designed for tight integration, they also can be used in more loosely coupled ways that should also be considered. It is also a good practice to leverage an I/O middleware library such as ADIOS, GLEAN, or HDF5 to facilitate the loose coupling.[8,11]

Simple Versus Expressive

Mature visualization packages like VTK, ParaView, and VisIt contain hundreds of visualization operations that can be combined in numerous ways. ParaView Catalyst and VisIt libsim both provide extensive scripting capabilities to establish and manage the visualization. Although a powerful and expressive way to specify a visualization process, writing scripts can be challenging for users. Also, because the underlying visualization system does not know what components are needed until the script is parsed at runtime, the visualization system tends to load unnecessary components.

In contrast, if the simulation results can be communicated through a small set of "precanned" visualizations or through a limited number of parameters, then the visualization can be specified in a much more user-friendly way, usually as a short specification in the simulation's input deck. Of course, a simplified interface limits the visualization system's abilities, and feature creep will eventually lead to a more complicated interface.

As with all of these tensions, there is no perfect answer. ParaView Catalyst tries to be flexible enough to support whatever level of complexity and expressiveness is appropriate for a given simulation. In addition to the fully expressive scripting that allows any configuration to be specified at runtime, the Catalyst library also features a C++ interface on which a simulation can build a much simpler interface. Catalyst Editions allow developers to subselect features appropriate for a particular simulation, and the ParaView GUI has a tracing capability that can automatically build scripts based on the simpler GUI interaction.

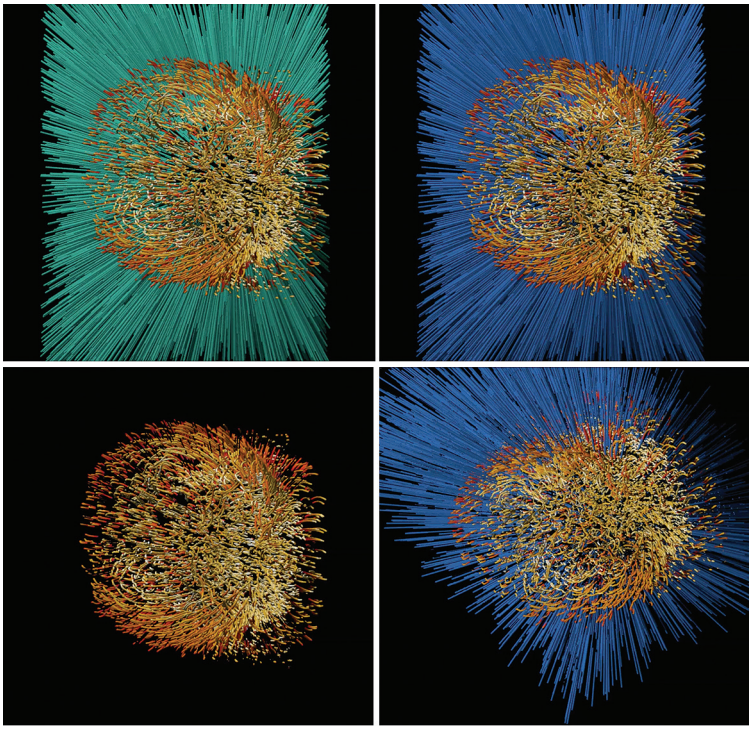

Figure 3. Example use of an explorable image of path tracing. This specially rendered image can later be manipulated to change material properties like color, remove some features, and make limited camera movements.

Small, Custom Versus Large, Feature-Rich

Over the past two decades there has been a large investment in scientific visualization for high-performance computing. This R&D has led to feature-rich visualization systems such as ParaView and VisIt that contain hundreds of visualization operations. As such, there is a strong motivation to leverage this software development for in situ usage, and libraries such as Catalyst for ParaView and libsim for VisIt do just that.

The trouble is that these large visualization libraries come with a lot of baggage. Because they are so feature rich, the libraries can add hundreds of megabytes to a simulation's executable alone. Because the instructions in the executable must be replicated for each process in a message passing interface (MPI) job (hundreds of thousands or more for large jobs), this adds to a substantial

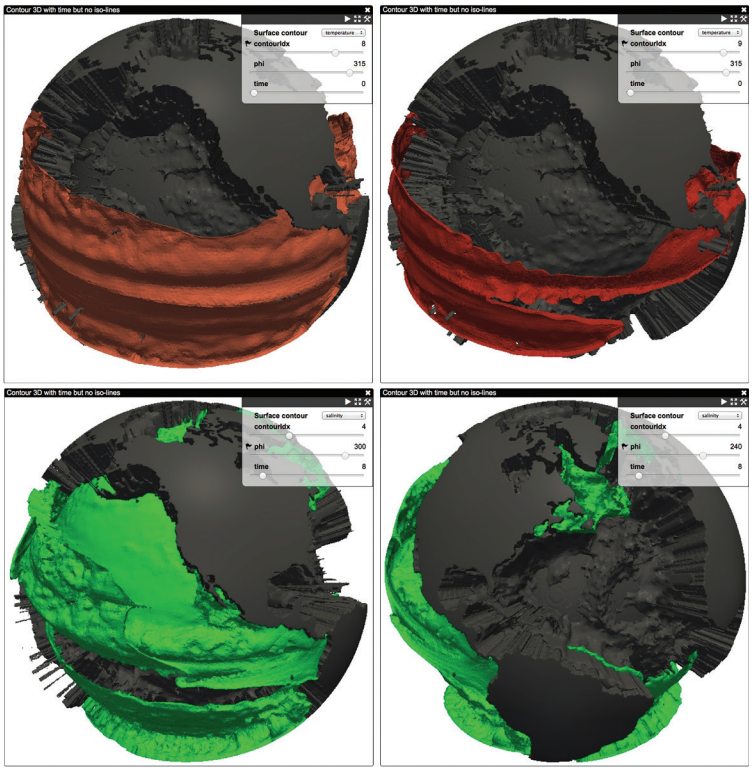

Figure 4. Example of using a Cinema image database. Each image is annotated with visualization parameters, so the Cinema application can browse images by manipulating visualization parameters like isovalues, camera position, and time in a manner similar to post hoc visualization. (Courtesy of Patrick O'Leary, Kitware)

amount of memory usage. Also, both Catalyst and libsim have complicated build systems that can be challenging for simulation developers, particularly when compiling for bleeding-edge computers. The complexity of these large software packages is also intimidating to software teams intent on robustness. Therefore, simulation groups may be hesitant to adopt these existing feature-rich libraries.

On the opposite end of the spectrum is a small, custom, single-purpose routine embedded in simulation code. This simple code has a low overhead and is easy to maintain. However, in addition to the obvious problem of code duplication, this simple visualization function can suffer from feature creep as users continually demand more from the visualization.

Our team for the Catalyst library helps alleviate this tension with what we call Catalyst Editions. An Edition consists of a subset of the overall Catalyst software, which can be customized to contain only the features needed by the simulation. Using a smaller Edition reduces the amount of memory needed for the program instructions and can greatly simplify the build process. Of course, the abilities of the in situ visualization are limited to what is included in the Edition, but new features can be added without duplicating the implementation. Although an Edition will not be as small and simple as a custom-built routine can be, it provides a good compromise to alleviate some of this tension.

Effective in situ visualization is not achieved by simply mashing together simulation and visualization software. Rather, there is a delicate balance of tradeoffs between mutually exclusive design goals and constraints, which creates tensions in the design and execution of in situ visualization systems.

It is often easy to see only one side or the other of these opposing requirements, but to design a successful in situ visualization system, it is important to understand all the concerns and be flexible. Rarely does one size fit all, so when providing a general-purpose utility, it is necessary to account for different use cases and operating modes.

Acknowledgments

This article draws on great work from many more individuals than I can possibly thank here. I would particularly like to thank W. Alan Scott, Nathan Fabian, Jeffrey Mauldin, and David Karelitz from Sandia National Laboratories; Andy Bauer, Utkarsh Ayachit, Berk Geveci, and Patrick O'Leary from Kitware; and James Ahrens and John Patchett from Los Alamos National Laboratory. This material is based in part on work supported by the US Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, under award numbers 14-017566 and 12-015215. This work was also funded by the US Department of Energy, National Nuclear Security Administration, Advanced Simulation and Computing Program. Sandia National Laboratories is a multiprogram laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin, for the US Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

References

1. S. Ahern et al., *Scientific Discovery at the Exascale: Report from the DOE ASCR 2011 Workshop on Exascale Data Management, Analysis, and Visualization*, Office of Advanced Scientific Computing Research, 2011; http://science.energy.gov/~/media/ascr/pdf/program-documents/docs/Exascale-ASCR-Analysis.pdf.
2. K. Moreland, "Oh, $#*@! Exascale! The Effect of Emerging Architectures on Scientific Discovery," *Proc. 2012 SC Companion: High Performance*

Computing, Networking Storage and Analysis (SCC), 2012, pp. 224–231.
3. A.G. Landge et al., "In-Situ Feature Extraction of Large Scale Combustion Simulations Using Segmented Merge Trees," *Proc. Int'l Conf. High Performance Computing, Networking, Storage and Analysis* (SC), 2014, pp. 1020–1031.
4. B. Nouanesengsy et al., "ADR Visualization: A Generalized Framework for Ranking Large-Scale Scientific Data Using Analysis-Driven Refinement," *Proc. 4th IEEE Symp. Large Data Analysis and Visualization*, 2014, pp. 43–50.
5. A. Tikhonova, C.D. Correa, and K.-L. Ma, "Visualization by Proxy: A Novel Framework for Deferred Interaction with Volume Data," *IEEE Trans. Visualization and Computer Graphics*, vol. 16, no. 6, 2010, pp. 1551–1559.
6. Y. Ye, R. Miller, and K.-L. Ma, "In Situ Pathtube Visualization with Explorable Images," *Proc. 13th Eurographics Symp. Parallel Graphics and Visualization* (EGPGV), 2013, pp. 9–16.
7. J. Ahrens et al., "An Image-Based Approach to Extreme Scale In Situ Visualization and Analysis," *Proc. Int'l Conf. High Performance Computing, Networking, Storage and Analysis* (SC), 2014, pp. 424–434.
8. K. Moreland et al., "Examples of In Transit Visualization," *Proc. 2nd Int'l Workshop Petascale Data Analytics: Challenges and Opportunities*, 2011, pp. 1–6.
9. M. Gamell et al., "Exploring Power Behaviors and Trade-offs of In-situ Data Analytics," *Proc. Int'l Conf. High Performance Computing, Networking, Storage and Analysis* (SC), 2013, article no. 77.
10. R. Oldfield et al., "Evaluation of Methods to Integrate Analysis into a Large-Scale Shock Physics Code," *Proc. 28th ACM Int'l Conf. Supercomputing* (ICA), 2014, pp. 83–92.
11. J.C. Bennett et al., "Combining In-situ and In-transit Processing to Enable Extreme-Scale Scientific Analysis," *Proc. Conf. High Performance Computing, Networking, Storage and Analysis* (SC), 2012, article no. 49.

Kenneth Moreland *is a principal member of the technical staff at Sandia National Laboratories. Contact him at kmorel@sandia.gov.*

Contact department editor Theresa-Marie Rhyne at theresamarierhyne@gmail.com.

cn Selected CS articles and columns are also available for free at http://ComputingNow.computer.org.

Keeping YOU at the Center of Technology

IEEE Computer Society Publications

Stay Informed

Access to Computer Society books, technical magazines and research journals arm you with Industry intelligence to keep you ahead of the learning curve.

- 3,000 technical books included with membership from books 24 x 7 and Safari Books Online
- 13 technical magazines
- 20 research journals

Learn something new. Check out Computer Society publications today!

Stay relevant with the IEEE Computer Society

More at www.computer.org/publications

IEEE computer society