VisIt: An End-User Tool for Visualizing and Analyzing Very Large Data

Hank Childs

Computational Research Division
Lawrence Berkeley National Laboratory,
Berkeley, California, USA, 94720.

Eric Brugger

Lawrence Livermore National Laboratory,
Livermore, CA, USA, 94551.

Brad Whitlock

Lawrence Livermore National Laboratory,
Livermore, CA, USA, 94551.

Jeremy Meredith

Oak Ridge National Laboratory,
Oak Ridge, TN, USA, 37831.

Sean Ahern

Oak Ridge National Laboratory,
Oak Ridge, TN, USA, 37831.

David Pugmire

Oak Ridge National Laboratory,
Oak Ridge, TN, USA, 37831.

Kathleen Biagas

Lawrence Livermore National Laboratory,
Livermore, CA, USA, 94551.

Mark Miller

Lawrence Livermore National Laboratory,
Livermore, CA, USA, 94551.

Cyrus Harrison

Lawrence Livermore National Laboratory,
Livermore, CA, USA, 94551.

Gunther H. Weber

Computational Research Division
Lawrence Berkeley National Laboratory,
Berkeley, California, USA, 94720.

Hari Krishnan

Computational Research Division
Lawrence Berkeley National Laboratory,
Berkeley, California, USA, 94720.

Thomas Fogal

University of Utah,
Salt Lake City, UT, USA, 84112.

Allen Sanderson

University of Utah,
Salt Lake City, UT, USA, 84112.

Christoph Garth

University of Kaiserslautern,
Kaiserslautern, Germany.

E. Wes Bethel

Computational Research Division
Lawrence Berkeley National Laboratory,
Berkeley, California, USA, 94720.

David Camp

Computational Research Division
Lawrence Berkeley National Laboratory,
Berkeley, California, USA, 94720.

Oliver Rübel

Computational Research Division
Lawrence Berkeley National Laboratory,
Berkeley, California, USA, 94720.

Marc Durant

Tech-X Corporation,
Boulder, CO, USA, 80303.

Jean Favre

Swiss Center for Scientific Computing,
Lugano, Switzerland.

Paul Návratil

Texas Advanced Computing Center,
Austin, TX, USA, 78758.

October 2012

Acknowledgment

This work was supported by the Director, Office of Science, Office and Advanced Scientific Computing Research, of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231. Some of the research in this work used resources of the National Energy Research Scientific Computing Center, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.

Legal Disclaimer

Abstract

VisIt is a popular open source tool for visualizing and analyzing data. It owes its success to its foci of increasing data understanding, large data support, and providing a robust and usable product, as well as its underlying design that fits today's supercomputing landscape. This report, which draws heavily from a publication at the *SciDAC Conference* in 2011 by Childs et al. [3], describes the VisIt project and its accomplishments.

Preface

The material in this technical report is a chapter from the book entitled *High Performance Visualization—Enabling Extreme Scale Scientific Insight* [1], published by Taylor & Francis, and part of the CRC Computational Science series.

Contents

1 Introduction

A dozen years ago, when the VisIt project started, a new high performance computing environment was emerging. Ever increasing numbers of end users were running simulations and generating large data. This rapidly growing number of large data sets prevented visualization experts from being intimately involved in the visualization process; it was necessary to put tools in the end users' hands. Almost all end users were sitting in front of high-end desktop machines with powerful graphics cards. But their simulations were being run on remote, parallel machines and generating data sets too large to be transferred back to these desktops. Worse, these data sets were too large to even process on their (serial) machines anyways. The types of visualization and analysis users wanted to perform varied greatly; users needed many techniques for understanding diverse types of data, with use cases ranging from confirming that a simulation was running smoothly to communicating the results of a simulation to a larger audience, to gaining insight via data exploration.

VisIt was developed in response to these emerging needs. It was (and is) an open source project for visualizing and analyzing extremely large data sets. The project has evolved around three focal points: (1) enabling data understanding, (2) scalable support for extremely large data, and (3) providing a robust and usable product for end users.

In turn, these focal points have made VisIt a very popular tool for visualizing and analyzing the data sets generated on the world's largest supercomputers. VisIt received a 2005 R&D 100 award for the tool's capabilities in understanding large data sets. It has been downloaded hundreds of thousands of times, and it is used all over the world.

2 Focal Points

2.1 Enable Data Understanding

In many ways, "VisIt" is a misnomer, as the name implies the tool is strictly about visualization and making pretty pictures. The prospect of lost name recognition makes renaming the tool unpalatable, but it is worthwhile to emphasize that VisIt focuses on five primary use cases:

1. *Visual exploration*: users apply a variety of visualization algorithms to "see" what is in their data.
2. *Debugging*: users apply algorithms to find a "needle in a haystack," for example, such as hot spots in a scalar field or cells that have become twisted over time. The user then asks for debugging information in a representation that is recognizable to their simulation (e.g., cell **X** in computation domain **D** has a NaN).
3. *Quantitative analysis*: users apply quantitative capabilities ranging from simple operations, such as integrating densities over a region to find its mass, to highly sophisticated operations, such as adding a synthetic diagnostic to compare to experimental data.
4. *Comparative analysis*: users compare two related simulations, two time slices from a single simulation, simulation and experiment, etc. The taxonomy of comparative analysis has three major branches, each of which is available in VisIt: image-level comparisons place things side-by-side and has the user detect differences visually. Data-level comparisons put multiple fields onto the same mesh, for example, to create a new field for further analysis that contains the difference in temperature between two simulations. Topological-level comparisons detect features in the data sets and then allow those features to be compared.
5. *Communication*: users communicate properties of their data to a large audience. This may be via movies, via images that are inserted into a PowerPoint presentation, or via line plots or histograms that are placed into a journal article.

2.2 Support for Large Data

Twelve years ago, "large data" meant several hundred million cells. Today, "large" means several hundred billion cells. In both cases, the definition of "large" was relative to the resources for processing the data. And this is the target for the VisIt project: data whose full resolution cannot fit into primary memory of a desktop machine. Of course, the amount of data to load varies by situation. Can time slices be processed one at a time? How many variables are needed in a given analysis? Is it necessary to load multiple members of an ensemble simultaneously? For VisIt, the goal was to provide an infrastructure that could support any of these use cases, and it primarily uses parallelism to achieve this goal.

2.3 Provide a Robust and Usable Product for End Users

Enabling data understanding for large data is a daunting task requiring a substantial investment. To amortize this cost, the project needed to be delivered to many user communities, across both application areas and funding groups.

The "one big tool" strategy provides benefits to both users and developers. Compared to a smaller, tailored effort, users have access to more functionality and better underlying algorithms for processing data. For developers, the core infrastructure undergoes an economy of scale, where many developers can collectively develop a superior core infrastructure than they would be able to do independently.

But the "one big tool" approach has negative aspects as well. Their user interface tends to provide an overly rich interface where users find many features to be meaningless and simply view them as clutter. Further, developers must deal with a less nimble code base where making functionality changes sometimes leads to unexpectedly large coding efforts.

Further, delivering a product to a large end user community incurs significant cost in and of itself: the VisIt project has almost a thousand pages of manuals, several thousand regression tests that run every night, a sophisticated build process, and a variety of courses designed to teach people to how to use the tool. It requires multiinstitutional coordination for release management, for responses to user requests, and for software development. And, of course, the source code itself must be well documented to reduce barriers to entry for new developers.

The developers of the VisIt project decided to "go big": to pay the costs associated with large user and developer bases in the hopes of writing a tool that would be usable by many and developed by many.

3 Design

This section describes several facets of VisIt's design, including VisIt's architecture, its parallelism approach, and its user interface concepts.

3.1 Architecture

VisIt employs a client-server design, where both client and server are composed of multiple programs (see Fig. 1). Client-side programs, typically run on the user's local desktop, are responsible for both user interface and rendering, since interactivity is paramount for these activities. The client-side programs are:

- *gui:* A graphical user interface built using the Qt widget set.
- *cli:* A command line user interface built using the Python language.
- *viewer:* A program responsible for the visual display of data.
- Custom, streamlined user interfaces can also be added to VisIt. The interfaces can either complement the gui and cli or replace them altogether.

Server-side programs, typically run on a remote supercomputer that can access the user's data in a parallel fashion, are responsible for processing data. The server-side programs are:

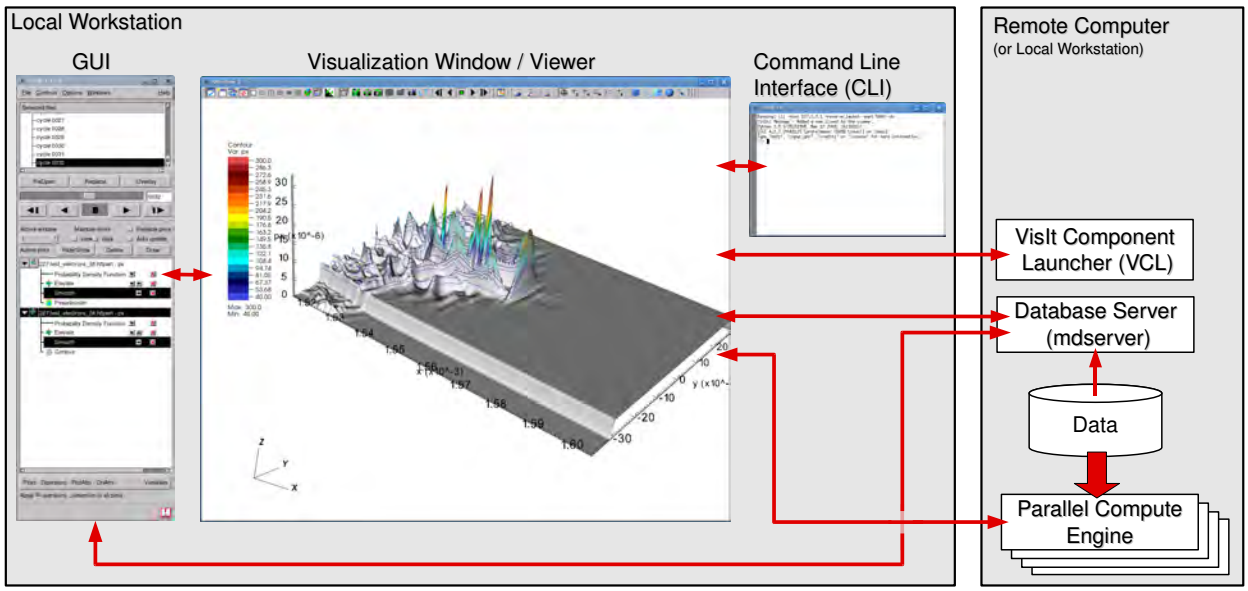

Figure 1: Diagram of VisIt programs and their communication. Image source: Childs et al., 2011 [3].

- *engine:* The program that applies visualization and analysis algorithms to large data sets using parallel processing.
- *mdserver:* A program that browses remote file systems and reads meta-data.
- *vcl:* **V**isIt **C**omponent **L**auncher, a program whose sole job is to launch other server-side programs. Without this program, the user would have to issue credentials for the launch of each program on the remote machine.

While the configuration in Figure 1 is the most common, other variants are also used:

- Data is located on the local machine, so all programs, including the server-side programs, run on the local machine.
- The client-side programs run on a remote machine. This mode occurs most often in conjunction with graphical desktop sharing, such as VNC.
- Multiple servers are run simultaneously to access data on multiple remote machines.
- VisIt is run entirely in "batch mode." The *gui* program is not used and the *viewer* program runs in a windowless mode.
- VisIt's client-side programs are coupled with a simulation code and data is processed *in situ*. In this case, the simulation embeds a copy of the *engine* program.

3.2 Parallelism

VisIt has multiple processing modes—multiresolution processing, *in situ* processing, and out-of-core processing—but its most frequent mode is pure parallelism, where the data is partitioned over its MPI tasks and is processed at its native resolution. Most visualization and analysis algorithms are embarrassingly parallel, meaning that portions of the data set can be processed in any order and without coordination and communication. For this case, VisIt's core infrastructure manages the partitioning of the data and all parallelism. For non-embarrassingly parallel cases like streamline calculation or volume rendering, algorithms are able to manage parallelism themselves and can opt to perform collective communication if necessary.

In VisIt's most typical visualization use case, a user loads a large data set, applies operations to reduce the data size, and then transfers the resulting data set to the local client for interactive rendering, using the local graphics card. However, some data sets are so large that their reduced

forms are too large for a desktop machine. This case requires a backup plan. VisIt's backup plan is to switch to a parallel rendering mode: data is left on the parallel server, each MPI task renders its own piece, and the resulting subimages are composited together. The final image is then brought back to the *viewer* and placed in the visualization window, as if it was rendered with the graphics card. Although this process sounds cumbersome, the switch to the parallel rendering mode is transparent to end users and frame rates approaching ten frames per second can be achieved.

VisIt was designed for many scales of concurrency. Many users run serial or modestly parallel versions on their desktop machines. When users utilize parallel resources on a supercomputer, they typically run with 32 to 512 tasks. But, for the largest data sets, VisIt servers with thousands or even tens of thousands of tasks are used (see Section 4.1). VisIt demonstrates excellent scalability and performance at each of these scales.

3.3 User Interface Concepts and Extensibility

Type	Description	# of instances
Database	How to read from a file	˜115
Operator	How to manipulate data	˜60
Plot	How to render data	˜20
Expression	How to derive new quantities	˜190
Queries	How to extract quantitative and debugging information	˜90

Table 1: VisIt's five primary user interface concepts.

Table 1 shows the five primary user interface concepts in VisIt. A strength of these concepts is their interoperability. Each plot can work on data directly from a file (databases) or from derived data (expressions), and can have an arbitrary number of data transformations or subselections applied (operators). Once the key information is extracted, quantitative or debugging information can be extracted (queries) or the data can be rendered (plots). Consider an example: a user reads from a file (database), calculates the λ-2 metric for finding high vorticity (expressions), isolates out the regions of highest vorticity operators, renders it (plots), then calculates the number of connected components and statistics about them (queries).

VisIt makes it easy to add new types of databases, operators, and plots. The base infrastructure deals with these concepts as abstract types; it only discovers the concrete databases, operators, and plots instances at start-up, by loading them as plug-ins. Thus, adding new functionality to VisIt translates to developing a new plug-in. Further, VisIt facilitates the plug-in development process. It provides an environment for defining a plug-in and also performs code generation. The developer starts by setting up options for the plug-ins, and then VisIt generates attributes for storing the options, user interface components (Python, Qt, and Java), the plug-in bindings, and C++ methods with "dummy" implementations. The developer then replaces the dummy implementations with their intended algorithm, file reading code, etc.

3.4 The Size and Breadth of VisIt

Although only briefly discussed in this report, VisIt has an extensive list of features. Its ˜115 file format readers include support for many HDF5- and NetCDF-based formats, CGNS, and others, including generic readers for some types of binary and ASCII files. Its ˜60 operators include transformations (such as projections, scaling, rotation, and translation), data subsetting (such as thresholding and contouring), and spatial thresholding (such as limiting to a box or a plane), among many others. Its ˜90 queries allow users to get customizable reports about

specific cells or points, integrate quantities, calculate surface areas and volumes, insert synthetic diagnostics/virtual detectors, and much more. Its ˜190 expressions go well beyond simple math. For example, the user can create derived quantities like, "if the magnitude of the gradient of density is greater than this, then do this, else do that."

And many features do not fit into the five primary user interface concepts. There is support for positioning light sources, making movies (including MPEG encoding), eliminating data based on known categorizations (e.g., "show me only this refinement level" from an AMR mesh), and rendering effects like shadows and specular highlights, to name a few. In total, VisIt is approximately one and a half million lines of code.

Finally, VisIt makes heavy use of the Visualization ToolKit (VTK) [18]. This library contains an execution model, a data model, and many algorithms for transforming data. VisIt implements its own execution model, but the other two pieces form the foundation of VisIt's data processing. VTK's data model forms the basis of VisIt's data model, although VisIt provides support for mixed material cells, metadata for faster processing, and other concepts not natively supported by VTK. Further, VisIt uses the native VTK algorithm for many embarrassingly parallel visualization algorithms. In short, VTK has provided an important leverage to the VisIt project, allowing VisIt developers to direct their attention to the project's three main focal points.

4 Successes

The VisIt project has succeeded in multiple ways: by providing a scalable infrastructure for visualization and analysis, by populating that infrastructure with cutting-edge algorithms, by informing the limits of new hardware architectures, and, most importantly, by enabling successes for the tool's end users. A few noteworthy highlights are summarized in the subsections below.

4.1 Scalability Successes

A pair of studies were run in 2009 to demonstrate VisIt's capabilities for scalability and large data (see Fig. 2). In the first study, VisIt's infrastructure and some of its key visualization algorithms were demonstrated to support weak scaling. This demonstration led to be VisIt being selected as a "Joule code," a formal certification process by the US Office of Management and Budget to ensure that programs running on high-end supercomputers are capable of using the machine efficiently. In the second study, VisIt was scaled up to tens of thousands of cores and used to visualize data sets with trillions of cells per time slice. This study found VisIt itself to perform quite well, although overall performance was limited by the supercomputer's I/O bandwidth. Both studies are further described by Childs et al. [7].

4.2 A Repository for Large Data Algorithms

Many advanced algorithms for visualizing and analyzing large data have been implemented inside of VisIt, making them directly available to end users. Notable algorithms include:

- A novel streamline algorithm that melds two different parallelization strategies ("over data" and "over seeds") to retain their positive effects while minimizing their negative ones [14];
- A volume rendering algorithm that handles the compositing complexities inherent to unstructured meshes while still delivering scalable performance [5];
- An algorithm for identifying connected components in unstructured meshes in a distributed-memory parallel setting on very large data sets [10];
- An algorithm for creating crack-free isosurfaces for adaptive mesh refinement data, a common mesh type for very large data [19];

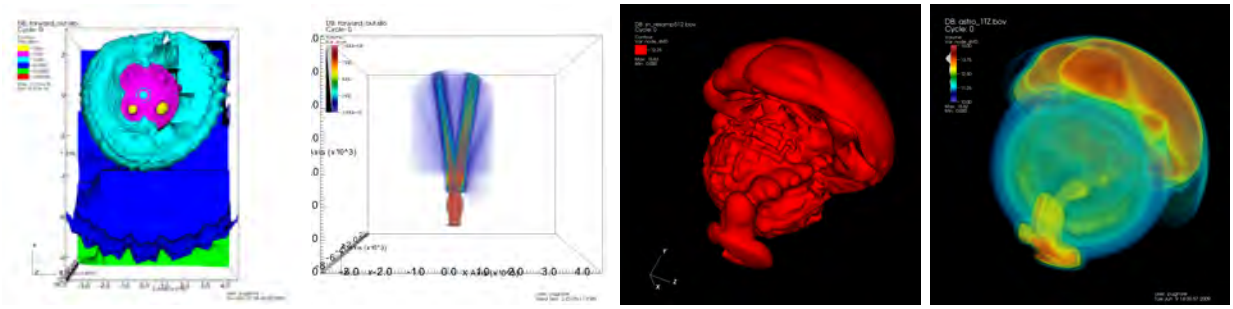

Figure 2: The two left images show a contouring and a volume rendering from a Denovo radiation transport simulation. They were produced by VisIt using 12,270 cores of JaguarPF as part of the "Joule code" certification, which showed that VisIt is weakly scalable. The two right images show a contouring and a volume rendering of a two trillion cell data set produced by VisIt using 32,000 cores of JaguarPF as part of a study on scalability at high levels of concurrency and on large data sets. The volume rendering was reproduced in 2011 on a one trillion cell version of the data set using only 800 cores of the TACC Longhorn machine. Image source: Childs et al., 2010 [7].

- A well-performing material interface reconstruction algorithm for distributed-memory parallel environments that balances concerns for both visualization and analysis [13]; and
- A method for repeated interpolations of velocity fields in unstructured meshes, to accelerate streamlines [9].

Further, VisIt has been the subject of much systems research, including papers on the base VisIt architecture [6], VisIt's "contract" system which allows it to detect the processing requirements for the current operations and adaptively apply the best optimizations [4], and a description of the adapter layer that allows VisIt to couple with a simulation and run *in situ* [20].

4.3 Supercomputing Research Performed with VisIt

As the landscape for parallel computers changes, VisIt has been used to test the benefits of emerging algorithms and hardware features, including:

- Studying modifications to collective communication patterns for ghost data generation, to be suitable for out-of-core processing, thereby improving cache coherency and reducing memory footprint [11];
- Studying the viability of hardware-accelerated volume rendering on distributed-memory parallel visualization clusters powered by GPUs [8];
- Studying the benefits of hybrid parallelism for streamline algorithms [2]; and
- Studying the issues and strategies for porting to new operating systems [15].

4.4 User Successes

Of course, the most important measure for the project is helping users better understand their data. Unfortunately, metrics of success in this space are difficult:

- Some national laboratories keep statistics on their user communities: the United States' Lawrence Livermore Lab has approximately 300 regular users, the United Kingdom's Atomic Weapons Establishment (AWE) has approximately 100 regular users, and France's Atomic Energy Commission (CEA) at CESTA has approximately 50 regular users. Other institutions, like Oak Ridge and Lawrence Berkeley, view VisIt as their primary visualization and analysis tool, but do not keep user statistics.

- In terms of monetary support for developing VisIt, the U.S. Department of Energy funds VisIt development through its Office of Science, National Nuclear Security Agency, and Office of Nuclear Energy. Both of the US National Science Foundation (NSF) XD centers on visualization actively deploy and support VisIt as well.
- Another method for measuring usage is studying affiliations of users who ask questions on the mailing list. The majority of these inquiries come from none of the previously mentioned institutions, indicating that usage goes beyond these sites.

Figure 3: Recent covers of the SciDAC Review Journal created using VisIt.

Tracking individual user successes is difficult, although there is clear evidence with certain types of usage. VisIt is used regularly to make images for journal covers, a high-profile activity (see Fig. 3). Further, there have been several notable instances of publications using VisIt to perform novel analysis:

- Analysis of laser wakefield simulations often amounts to finding key particles [16], and query-driven visualization techniques were used to search through terabytes of data to locate these key particles in as little as two seconds.
- Simulations often deal with idealized meshes. VisIt's comparative capabilities were used to quantify the importance of engineering defects when differencing as-built and as-designed models [12].
- VisIt's streamline code was used to find the toroidal magnetic fields found in tokamaks by analyzing the fieldlines through a cross-sectional slice and the topological "islands" they trace out [17].

5 Future Challenges

Although VisIt is well suited for today's supercomputing environment, the project will face many challenges in the future. In the short term, I/O limitations will force visualization and analysis activities to de-emphasize I/O. The VisIt development team has invested in pertinent techniques, such as multiresolution processing and *in situ*, but these techniques will need to be further hardened to support production use. In the longer term, power limits will constrain data movement, forcing much processing to occur *in situ* on novel architectures, such as GPU accelerators. Unfortunately, VisIt's existing *in situ* implementation may be mismatched for this many-core future, for two reasons. First, although VisIt can be easily multithreaded, using a pthreads or OpenMP-type approach to further understand the benefits of hybrid parallelism), this approach may not be able to take advantage of these architectures. The many-core future may require CUDA- or OpenCL-type languages; migrating the VisIt code base to this setting would be a substantial undertaking. Second, although VisIt has been demonstrated to work well at high levels of concurrency, some of its algorithms involve large data exchanges. Although these algorithms perform well on current machines, they would violate the data movement constraints on future machines and would need to be redesigned.

6 Conclusion

The VisIt project's three focal points—understanding data, large data, and delivering a product—together form a powerful environment for analyzing data from HPC simulations. It is used in a variety of ways: it enables visualization scientists, computational code developers, and the physicists that run these codes to perform a broad range of data understanding activities, including debugging, making movies, and exploring data. The user interface portion of its design provides a powerful paradigm for analyzing data while the data processing portion of its design is well suited for big data. This, in turn, has led to many successes: in scaling up to high levels of concurrency and large data sizes, in providing a "home" for large data algorithms, in understanding how to best use supercomputers, and, most importantly, in helping users understand their data. Further, despite significant upcoming changes in supercomputing architecture, VisIt's future appears bright, as it enjoys vibrant user and developer communities.

References

[1] E. Wes Bethel, Hank Childs, and Charles Hansen, editors. *High Performance Visualization—Enabling Extreme-Scale Scientific Insight*. Chapman & Hall, CRC Computational Science. CRC Press/Francis–Taylor Group, Boca Raton, FL, USA, November 2012. `http://www.crcpress.com/product/isbn/9781439875728`.

[2] David Camp, Christoph Garth, Hank Childs, Dave Pugmire, and Kenneth I. Joy. Streamline Integration Using MPI-Hybrid Parallelism on a Large Multicore Architecture. *IEEE Transactions on Visualization and Computer Graphics*, 17:1702–1713, 2011.

[3] Hank Childs, Eric Brugger, Brad Whitlock, Jeremy Meredith, Sean Ahern, Kathleen Bonnell, Mark Miller, Gunther H. Weber, Cyrus Harrison, David Pugmire, Thomas Fogal, Christoph Garth, Allen Sanderson, E. Wes Bethel, Marc Durant, David Camp, Jean M. Favre, Oliver Rübel, Paul Navrátil, Matthew Wheeler, Paul Selby, and Fabien Vivodtzev. VisIt: An End-User Tool For Visualizing and Analyzing Very Large Data. In *Proceedings of SciDAC 2011*, July 2011. `http://press.mcs.anl.gov/scidac2011`.

[4] Hank Childs, Eric S. Brugger, Kathleen S. Bonnell, Jeremy S Meredith, Mark Miller, Brad J Whitlock, and Nelson Max. A Contract-Based System for Large Data Visualization. In *Proceedings of IEEE Visualization*, pages 190–198, 2005.

[5] Hank Childs, Mark Duchaineau, and Kwan-Liu Ma. A Scalable, Hybrid Scheme for Volume Rendering Massive Data Sets. In *Proceedings of Eurographics Symposium on Parallel Graphics and Visualization*, pages 153–162, May 2006.

[6] Hank Childs and Mark Miller. Beyond Meat Grinders: An Analysis Framework Addressing the Scale and Complexity of Large Data Sets. In *SpringSim High Performance Computing Symposium (HPC 2006)*, pages 181–186, 2006.

[7] Hank Childs, David Pugmire, Sean Ahern, Brad Whitlock, Mark Howison, Prabhat, Gunther Weber, and E. Wes Bethel. Extreme Scaling of Production Visualization Software on Diverse Architectures. *IEEE Computer Graphics and Applications*, 30(3):22–31, May/June 2010.

[8] Thomas Fogal, Hank Childs, Siddharth Shankar, J. Krüger, R. D. Bergeron, and P. Hatcher. Large Data Visualization on Distributed Memory Multi-GPU Clusters. In *Proceedings of High Performance Graphics 2010*, pages 57–66, June 2010.

[9] Christoph Garth and Ken Joy. Fast, Memory-Efficient Cell Location in Unstructured Grids for Visualization. *IEEE Transactions on Computer Graphics and Visualization*, 16(6):1541–1550, November 2010.

[10] Cyrus Harrison, Hank Childs, and Kelly P. Gaither. Data-Parallel Mesh Connected Components Labeling and Analysis. In *Proceedings of EuroGraphics Symposium on Parallel Graphics and Visualization*, pages 131–140, April 2011.

[11] Martin Isenburg, Peter Lindstrom, and H. Childs. Parallel and Streaming Generation of Ghost Data for Structured Grids. *IEEE Computer Graphics and Applications*, 30(3):32–44, May/June 2010.

[12] Edwin J. Kokko, Harry E. Martz, Diane J. Chinn, Hank R. Childs, Jessie A. Jackson, David H. Chambers, Daniel J. Schneberk, and Grace A. Clark. As-Built Modeling of Objects for Performance Assessment. *Journal of Computing and Information Science in Engineering*, 6(4):405–417, 12 2006.

[13] Jeremy S. Meredith and Hank Childs. Visualization and Analysis-Oriented Reconstruction of Material Interfaces. *Computer Graphics Forum*, 29(3):1241–1250, 2010.

[14] D. Pugmire, H. Childs, C. Garth, S. Ahern, and G. Weber. Scalable Computation of Streamlines on Very Large Datasets. In *Proceedings of Supercomputing (SC09)*, 2009.

[15] David Pugmire, Hank Childs, and Sean Ahern. Parallel Analysis and Visualization on Cray Compute Node Linux. In *Proceedings of the Cray Users Group Meeting*, 2008.

[16] Oliver Rübel, Prabhat, Kesheng Wu, Hank Childs, Jeremy Meredith, Cameron G. R. Geddes, Estelle Cormier-Michel, Sean Ahern, Gunther H. Weber, Peter Messmer, Hans Hagen, Bernd Hamann, and E. Wes Bethel. High Performance Multivariate Visual Data Exploration for Extemely Large Data. In *Proceedings of Supercomputing (SC08)*, 2008.

[17] A. R. Sanderson, G. Chen, X. Tricoche, D. Pugmire, S. Kruger, and J. Breslau. Analysis of Recurrent Patterns in Toroidal Magnetic Fields. *IEEE Transactions on Visualization and Computer Graphics*, 16(4):1431–1440, 2010.

[18] William J. Schroeder, Kenneth M. Martin, and William E. Lorensen. The Design and Implementation of an Object-Oriented Toolkit for 3D Graphics and Visualization. In *Proceedings of the IEEE Conference on Visualization (Vis96)*, pages 93–100, 1996.

[19] Gunther H. Weber, Vincent E. Beckner, Hank Childs, Terry J. Ligocki, Mark Miller, Brian van Straalen, and E. Wes Bethel. Visualization Tools for Adaptive Mesh Refinement Data. In *Proceedings of the 4th High End Visualization Workshop*, pages 12–25, 2007.

[20] Brad Whitlock, Jean M. Favre, and Jeremy S. Meredith. Parallel In Situ Coupling of Simulation with a Fully Featured Visualization System. In *Proceedings of EuroGraphics Symposium on Parallel Graphics and Visualization*, pages 101–109, April 2011.