Research Challenges for Visualization Software

Hank Childs[1], Berk Geveci and Will Schroeder[2], Jeremy Meredith[3], Kenneth Moreland[4], Christopher Sewell[5], Torsten Kuhlen[6], E. Wes Bethel[7]

May 2013

[1]University of Oregon and Lawrence Berkeley National Laboratory
[2]Kitware, Inc.
[3]Oak Ridge National Laboratory
[4]Sandia National Laboratories
[5]Los Alamos National Laboratory
[6]RWTH Aachen University, Germany
[7]Lawrence Berkeley National Laboratory

Acknowledgment

This work was supported by the Director, Office of Science, Office of Advanced Scientific Computing Research, of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.

Legal Disclaimer

Research Challenges for Visualization Software

Hank Childs, Berk Geveci, Jeremy Meredith, Kenneth Moreland, Christopher Sewell, E. Wes Bethel, Torsten Kuhlen, and Will Schroeder

Abstract

Over the last twenty-five years, visualization software has evolved into robust frameworks that can be used for research projects, rapid prototype development, or as the basis of richly featured, end-user tools. In this article, we take stock of current capabilities and describe upcoming challenges facing visualization software in six categories: massive parallelization, emerging processor architectures, application architecture and data management, data models, rendering, and interaction. Further, for each of these categories, we describe evolutionary advances sufficient to meet the visualization software challenge, and posit areas in which revolutionary advances are required.

1 Introduction

Visualization is an enabling technology; its purpose is to create insight from data and it performs this task across many domains. It is an essential tool for exploring, confirming, and communicating trends in data, and its utility goes well beyond just that of "pretty pictures." The dominant mechanism for delivering visualization technology to end users is software – applications and libraries. Usage of visualization software is ubiquitous, with examples ranging from common information graphics to temperature maps on the nightly news weather report to intricate representations of complex physics. This software is effective because it allows the expertise of small developments teams to be deployed efficiently to large user groups. Further, although these user groups know how to interpret results, the typical consumer does not know how to create visualizations on their own; visualization software encapsulates the details of the techniques and hides complexity for these consumers by providing a set of canned visualization algorithms.

Visualization is evolving due to several major trends in computing. Big Data [Nor11] is increasingly driving software architectures. Such data not only brings voluminous content, but an accelerating diversity of data forms as well. For example, the term visualization refers to more than the spatiotemporal data forms associated with scientific visualization. With the advent of information visualization and analytics, visualization systems must also contend with unstructured data forms such as text, graphs, trees, tables and other metadata. Of course, Big Data also can refer to very high-resolution data sets and we consider both types throughout this article. In addition to Big Data, another important trend is the exploding diversity in parallel computing systems, rendering solutions, interaction devices, and delivery platforms such as tablets. This stresses the ability of software to represent, process and interact with data in efficient ways. And finally, the assumption that data can be readily transferred to and from storage is breaking down; power requirements and bandwidth limitations mean that visualization must move closer to the data to extract the most meaningful information efficiently.

This article surveys the research challenges facing visualization software today. While we primarily focus on the challenges for scientific visualization, we also touch on related challenges for information visualization in some places. The article is organized into discussions of general visualization topics: massive parallelization (§2), processor architecture (§3), application architecture and data management (§4), data models (§5), rendering (§6), and interactions (§7). Although some of the conversation borders on challenges in the topics themselves, we always shift the focus onto the software research challenges: how to modularize to isolate complexity, system design and delivery mechanisms, dealing with emerging computing and programming models, providing the right abstractions to facilitate algorithm development, and how to "future proof." Further, we conclude our discussion of each topic with an assessment of which challenges can be met with evolutionary changes and which require revolutionary ones.

Sidebar: Visualization Software: 1988-2013

Over the last twenty-five years, design patterns have emerged and standardized across a variety of visualization software packages. These design patterns have both simplified the development of visualization software and enabled

their user interfaces to efficiently present myriad features to users in a streamlined fashion. The most important example is that of breaking up visualization operations into modules and connecting these modules, in an arbitrary and often dynamic fashion, to create pipelines.

Visualization libraries form the basis of most visualization applications. Popular examples of these libraries are the Visualization ToolKit (VTK) [SML96], Advanced Visual Systems (AVS) [UJK*89], and Open Data Explorer (OpenDX) [AT95]. At their heart, visualization libraries typically provide three things: an execution model, a data model, and a collection of modules that input, process, and/or output data. Their design is deceptively elegant; they define natural interfaces that encapsulate functionality (meaning that software is centralized, isolated, and implemented only once). For example, the data model, which comprises the core memory structures capturing the nature of the data, is independent from both execution model and modules. The program execution strategy, which includes issues such as dependencies, caching, and load balancing, is totally separate from the modules that process data. The modules that process data – whether they are file format readers, filters that contain algorithms that transform data, or rendering modules – do not require any knowledge of the context in which they are executing. They simply know that they produce an output and what their inputs are. By defining boundaries between data model, modules, and execution, the job of implementing each of them is significantly less cumbersome. This division is particularly important because there are often hundreds to thousands of modules compared to a handful of execution strategies (or even just a single one) and one data model.

There are additional advantages to this design. First, it is highly extensible. New modules can be developed without modifying the core infrastructure. For example, readers that process a new file format can be added seamlessly to the library, because these readers produce output using the same data model that the filters consume. Second, the design allows for dynamic composition of modules. Stringing together multiple visualization operations is often useful – e.g., read data, then slice it, then trace particles along the slice, then remove the particle paths that do not travel far, then render the resulting long paths. As a result, visualization libraries tend to have modules that perform small, indivisible tasks that users can build up to produce the exact visualization they want.

Visualization applications often utilize visualization libraries; further, they provide an interface that allows users to combine modules and set their attributes to produce the desired results. A significant benefit of this approach is that application development costs are substantially reduced. In the simplest form, visualization applications merely contain user interface code and control code to set up and connect modules from the visualization library. Further, since the visualization libraries have many modules, applications that "buy in" to a library's infrastructure can pick up new modules at minimal costs, providing functionality developers could have deemed too expensive to implement otherwise. Another benefit of reduced development costs is that prototype applications can be easily constructed, often in less than a day or even through automated application builders.

Visualization applications have thousands of options; organizing them is a task no less daunting than that of the interface design for applications like PowerPoint™ or Photoshop®. Visualization applications often mirror the abstractions presented by visualization libraries as a way to organize these options, meaning they present users with a list of modules, attributes for a module, and module composition controls. The resulting interface gives the user a high level of control over how to carry out a visualization, and, surprisingly, is often deemed intuitive. VisIt [CBW*11] and ParaView [AGL05] (applications developed by the authors of this article) are popular examples that fit this model.

2 Massive Parallelization

Parallelization can occur both within a compute node and across compute nodes†. The parallelization techniques for these environments are distinct and we divide their treatment into two sections. This section focuses on the parallelism across multiple compute nodes, while §3 focuses on parallelism within a single compute node.

Due to increasing data sizes and the emergence of the Big Data problem, the need for massive parallelization is a driving visualization research challenge. Supercomputing simulations regularly generate massive data sets with billions of data points per time step. Parallelization is an effective way of dealing with such data: there is more memory for storing data, there is more compute power for executing algorithms, and there is often more I/O bandwidth for reading data. The basic challenge for parallel visualization algorithms is to decompose the problem into independent tasks that can be run concurrently on all of the **processing elements** (i.e., the instances of the program), thus avoiding idle time. Data parallelism is the dominant technique; data sets are decomposed into pieces and the pieces are partitioned over the processing elements. This approach has been shown to be highly scalable with results for hundreds of thousands of processing elements in research prototypes [HBC12] and tens of thousands of processing elements in production software [CPA*10].

The role of visualization software, with respect to parallelization, is to provide a framework that shields algorithm developers from complexity. Most commonly, this framework extends the same abstractions found in the traditional

† This article defines a **compute node** as group of cores that can share memory.

data flow design. Much in the same manner that data flow design hides execution details from algorithms in a serial setting, parallel visualization software can hide an even more complex story in a parallel setting, i.e., managing the decomposition, distribution, and collection of pieces, ensuring that artifacts don't occur along piece boundaries, etc. In short, a single investment in the framework can spare redundant implementations in a multitude of filters.

Upcoming Challenges:

As High Performance Computing (HPC) progresses towards the exascale (10^{18} floating point operations per second), today's parallelization approaches will face many challenges.

First, though the number of cores on a single supercomputer will likely exceed one billion, the growth in parallelism from today will mostly occur within a compute node. For example, visualization software may require as few as one million tasks at scale — one per compute node — and instead achieve parallelism within a node using techniques discussed in §3. Visualization software has already been run successfully with hundreds of thousands of processing elements, so exasale computers should lead to only an additional order of magnitude. Algorithms that parallelize well should require only modest improvements, while algorithms that are hard to parallelize today will get that much harder. The biggest challenge, however, will become how to manage the **hybrid parallelism** that blends between distributed- and shared-memory techniques. Visualization software will again need to provide infrastructure that both shields algorithm developers from complexities and also provides them the richness to process data in the most efficient way. Issues that must be addressed at the infrastructure level include work scheduling, fault tolerance, and efficient support for heterogeneous resources.

Second, the energy requirements of these machines will be a fundamental concern, and this will limit data movement. As a result, the traditional model of a visualization program acting as a post-processor (i.e., reading simulation results from disk after the simulation has finished) will be jeopardized, as the cost to regularly store snapshots of the simulation to disk will be prohibitive. Instead, visualization software will need to be transformed to operate in an *in situ* manner, where it either performs visualization directly or where it reorganizes and reduces data so that significantly smaller data sets can be stored for later processing. Of course, *in situ* processing also can have benefits, by enabling access to more data, by providing immediate feedback, and creating the possibility for steering capabilities. The software issues for *in situ* processing are discussed further in §4.

Third, data sizes will increase from billions of data points per time slice to trillions of data points per time slice. Parallelization should provide the necessary compute power to carry out algorithms on this massive data. However, secondary challenges emerge centering around data integrity and data understanding as data gets reduced to millions of pixels. Whatever new techniques to address these challenges, visualization software will need to provide infrastructure that supports it.

Finally, scientific visualization does not have a monopoly on Big Data. Information visualization data sets have the potential to increase at an even faster rate and will share many of the same challenges: how software can manage massive concurrency, how and where to process data, and how to represent data in a way that maintains integrity.

Evolutionary / Revolutionary: 75% / 25%

Although higher levels of parallelization will produce qualitatively new challenges, the fundamental elements of parallel visualization software provide a solid foundation for future extensions. The challenges for visualization software that have the most revolutionary potential will be managing a heterogeneous environment, delivering *in situ* solutions, and supporting new approaches for non-embarrassingly parallel algorithms.

3 Processor Architecture and Programming Model

Visualization software today most often is implemented in standard languages such as C and C++ with no specialized constructs for parallelism beyond the ability to pass messages between nodes (i.e. "MPI" programming). This is because, historically, improvements in CPU performance have been obtained through gains in single-threaded performance (i.e. a faster clock speed). The most common form of parallelism in visualization has been the form described in §2, via a distributed decomposition of the data. The processing after the decomposition is, in essence, serial.

In recent years, the energy costs for gains in single-threaded performance have become significant. As a result, CPU clock speeds have remained largely constant, but more cores are being placed on a single node. Software has been slow to keep pace and a common strategy is to create a processing element for each core. Opportunities for increased efficiency through reduced communication overhead and better synchronization have recently led to serious efforts employing true threading constructs for parallelism within a node. In these cases, the programming models have remained manageable; techniques like POSIX threads and OpenMP style compiler pragmas can be used within the same frameworks already familiar to visualization algorithm developers.

Unfortunately, the architectural changes are not limited to just modest increases in core counts. Some projections of future architectures have per-node concurrency exceeding a thousand-fold increase in this decade, leading to a different type of processor design with multiple consequences. Graphics processing units (GPUs), already successful for a variety of computational tasks and containing over one thousand lightweight cores, provide the best approximation of

these future architectures. Visualization software will face several challenges and require substantial investment.

Upcoming Challenges:

Visualization algorithms will be difficult to parallelize at a fine-grain level on these architectures. They depend more heavily on topology-based operations, such as information about neighboring elements, than many other algorithms. For example, at each cell a convolution filter requires information from all neighboring cells within the filter radius, and an isosurface requires information from every vertex of a cell, where vertices are shared with adjacent cells. Accessing such geometric connectivity information becomes even more complex in the context of data with no regular structure, such as those data forms found in information visualization such as graphs and trees.

Visualization algorithms often exhibit different access patterns than simulations running on the same hardware. This is because visualization algorithms typically exhibit a greater dependence on memory movement for their performance than computational horsepower. For example, an isosurface operator may only involve a few linear interpolations per cell, and a convolution operator only a weighted average computation, compared to a typical physics simulation that utilizes computation-intensive operations such as iterative non-linear solvers. Achieving good performance with memory accesses on highly parallel architectures is difficult, requiring investigation of strided access patterns, memory coalescing, bank conflicts, alignment and padding, and even exploring the use of built-in vector types for memory access.

As the variety of multi-core and accelerator hardware designs continues to increase, the visualization software developer is faced with the increasingly daunting challenge of re-optimizing or even re-writing his or her algorithms for different architectures, or even preparing for unknown future hardware. The variety of low-level programming models is also increasing, including options such as CUDA, OpenCL, and OpenACC. And while some are cross-platform standards, allowing the same code to compile and run on platforms as divergent as CPUs and GPUs, they do not ensure good performance on different platforms without platform-specific tuning and optimizations, as they do not attempt to solve the underlying issue that different algorithms, memory access patterns, etc. perform better on different architectures. This challenge is not unique to visualization software, but a variety of techniques are well suited to mitigating these issues in visualization frameworks. One such technique is the development of cache-oblivious algorithms, which prevent the algorithm from having to be tuned to the differing characteristics of each architecture's cache. Another technique is to design algorithms with very fine threading (e.g., millions of lightweight threads), which is far beyond what current architectures require, but increases their portability to different core counts and arrangements. Finally, we note that programming expertise in fine-grained parallelism is not common among veteran visualization software developers; as universities ramp up on teaching these skills, our community may struggle to find the personnel who can implement the next generation of visualization software.

Evolutionary / Revolutionary: 10% / 90%

Although it is true that the current generation of production visualization software is written in the wrong language for future architectures, the larger problem is that their algorithms must be re-thought to make use of fine-grained parallelism and optimize for performance across diverse architectures. The resulting frameworks must incorporate programming models that provide levels of abstraction that current frameworks lack. They must allow developers to encapsulate away idiosyncrasies of particular processor architectures and achieve portable performance. Ideally, these frameworks will go beyond array manipulations and provide support for a robust data model with a structural framework that optimizes data access patterns, as well as common visualization operations like field interpolations, adjacency relationships, topology generation, coincident point resolution, and cell finding. There are multiple attempts underway at building new frameworks that apply this approach to visualization software and the authors of this article are involved with several: DAX [MAGM11], EAVL [MAPS12], and PISTON [LSA12].

4 Application Architecture and Data Management

Application architecture refers to the system design of visualization software. Data management for visualization must provide visualization techniques that integrate into the data life cycle. Although these two topics are distinct, they are treated together here, since emerging data management needs will drive application architecture.

Traditionally, data management has not been a pressing concern for visualization software. Data, whether observed or simulated, was stored in the file system for processing; visualization software simply read whatever data it needed from files whenever it needed it. However, increases in data size, observed and simulated, as well as diversity of data sources, mandate new approaches in data management.

Application architectures exist to solve the simplest use model: "have data, want a picture," where the application architecture serves as a black box that consumes data and produces imagery with user-selected methods and parameters. Twenty-five years ago, the architecture for most visualization applications was a single binary that read from the local file system and produced images using local graphics. A little over a decade ago, scientific visualization applications for large data shifted to a client-server design where data was processed by a remote parallel server, producing geometry that was rendered by a local client. Today, application architectures for visualization frequently involve web clients and

remote data access. In short, application architectures evolve to meet evolving data management needs.

Upcoming Challenges:

The Big Data explosion happening worldwide directly affects visualization software and this software must evolve to fit within the data management ecosystem. The goal must be flexible, lightweight packages that can be integrated in a variety of delivery scenarios. For example, consider the experimental data measured by sensors. In many areas, this data is increasing dramatically both temporally and in fidelity. As a result, the data often can not be stored and sometimes can not even be transferred to compute-heavy resources, due to network limitations. The common strategy for this scenario is data triage; data is transformed and reduced as it comes off the device. A similar problem is occurring with simulated data for high-end supercomputers. As discussed in §2, *in situ* processing will be necessary to deal with power constraints. The best way to carry out this processing is unknown. It may involve direct incorporation into the simulation code, sending data to nearby resources where they are processed, or some combination of the two. Whatever the form, visualization software will need to be cognizant of the larger system in which it operates, as consuming inordinate amounts of memory or compute time can impact the simulation's performance. Consider an example: simulation data follows a fixed layout, i.e., their arrays are column-major or row-major, the components of their vector data are interleaved or not, etc. Visualization software often fixes the data layout it considers; when encountering simulation data that follows a different data layout, it copies the simulation's data into its familiar layout, using additional memory. Instead, the software must be able to perform "zero-copy" *in situ* that adapts to the simulation code's layout by using templates or virtual function calls. Thinking about the bigger picture, the common theme to these examples is that software must be designed to be lightweight and "run anywhere." Further, the associated complexities need to be abstracted away so that algorithm developers can focus solely on the details of their algorithm.

A second challenge comes from diverse sources of data. Successful analysis of observed data increasingly requires data fusion (i.e., merging multiple inputs). This may include classic scientific data forms, as well as unstructured metadata that may represent data outside of the spatiotemporal coordinate frame. Further, simulations now occur on multiple length-scales, and visualization is required to see how these length-scales interact. For both examples, visualization software must be able to deal with heterogeneous inputs from multiple sources and integrate them into in a single output. These sources may come from the local file system, from a remote database, or even from the cloud. The challenge for visualization software is to create systems that will manage multiple data management paradigms and isolate their associated complexities.

Finally, a shift in the computational landscape, namely the ubiquitous use of web- and cloud-based resources and delivery vehicles ranging from traditional desktop to web browser to tablet to smart phone, offers new challenges and opportunities not present in the early days of visualization application design and development. The main challenge here focuses on abstracting key interfaces, such as a rendering interface, in order to be able to take advantage of different rendering platforms and delivery vehicles. Further, it will be increasingly commonplace for a single visualization application to deploy on multiple platforms (e.g., web, tablet, etc.), including possible simultaneous usage (i.e., coordinating powerwall usage with a tablet); software infrastructure needs to insulate algorithms from the details of where it is being delivered so code can be delivered in multiple platforms simultaneously.

Evolutionary/Revolutionary: 80% / 20%

Although the challenges from Big Data will require further innovation, current visualization software design already has many lightweight and flexible aspects that will serve as a foundation. The main challenges encountered in data management touch on this article's other topics (§3, §6, §7).

5 Data Model

Visualization software is intended to be used with a wide range of data types. Scientific visualization tools must support data output from simulations in a disparate range of scientific domains, such as climate, fusion, and cosmology. Though this includes a wide range of application codes, they share many features and have a high degree of commonality in their needs. For example, general-purpose scientific visualization libraries have been able to import and analyze results from these codes by supporting structured and irregular finite element grids, from one to three spatial dimensions, with field data on the nodes and cells of the grids. Information visualization tools have faced a bigger challenge in supporting the large variety of structured and unstructured data encountered in informatics: from simple tables to complex graphs to unstructured collection of data samples. However, a few visualization libraries have been able to address a larger subset of information visualization challenges by providing relatively simple data models together with flexible programming interfaces to mold the data structure to fit a variety of problems.

Upcoming Challenges:

One significant challenge to data models in existing visualization software is that the types of data they are expected to handle is growing. In the scientific space, new refinement structures, new types of polynomial fields, and even high dimensional grids are becoming more common. The need for these structures are motivated both by the demands of new science and by the evolution of scientific computing algorithms.

Expectations for visualization software data type support are also growing because the demands on their analysis capabilities are growing. For example, general-purpose visualization software, which historically supported only continuum grids, might now be expected to handle results from particle codes. Furthermore, as scientists expand their toolbox to include emerging analytics techniques, they deal more and more with non-spatial data that is best explored by information visualization techniques. As users commonly expect to handle all this data and the associated visualization algorithms within a single visualization tool or library, its data model must be a superset of a vast array of other data models.

Other significant challenges in the area of data models are forced by coming changes in system architectures. The most obvious of these is that while we expect core counts to rise drastically, total system memory will increase only modestly; this results in a massive reduction in per-core memory. This means visualization libraries must be creative in finding new ways of storing the same data. In some cases, this is as straightforward as reducing redundancy inherent in a less descriptive data model. One example seen in today's software is a subset of a regular grid; in some data models, this can be achieved only via conversion to a much more expensive unstructured grid. A more flexible data model would allow hybrids of regular and explicit coordinates resulting in a more memory efficient representation critical for a memory-constrained future.

Another change in coming system architectures is the addition of heterogeneous processing units and many-core devices, as discussed in §3. Although programming these devices is itself a challenge, some aspects of this challenge must fall on the data model. The heterogeneity inherent in using discrete accelerator devices is one example; the underlying data structures should have support for these discrete memory spaces, or else accessing simulation data *in situ* may multiply memory usage in an already highly memory constrained scenario. These many-core devices, because of their need to hide memory latency through massive parallelism, are more sensitive to array layout than traditional cores, and in fact, have only limited caching at best. If the data model requires strided arrays for coordinates, for example, then it is enforcing a data layout that will preclude optimal performance on these devices.

The biggest software challenge associated with these emerging requirements is maintaining a programming interface that enables algorithm developers to produce general and special purpose algorithms easily while hiding the complexity of the underlying data structures. Recent history shows that the most successful visualization libraries have been those that make it easy for programmers to develop and maintain algorithms that work on a large set of data types. Next generation visualization software must maintain this property while supporting a growing set of data types, programming models, and target architectures.

Evolutionary / Revolutionary: 75% / 25%

The challenges of supporting new data models may require intrusive changes to existing software, and opportunities to optimize data structures to minimize memory usage and improve support for coming architectures are vast. However, there is no single step that appears to be revolutionary. Indeed, attempts at revolutionary data models risk becoming buried in mathematical quandaries and never achieving practicality. Instead, a number of evolutionary changes to existing data models are more likely to balance benefits with pragmatism.

6 Rendering

The rendering system is the bridge between the human observer and data. As such, it must map large, disparate, often complex data forms into graphics primitives (e.g., simple geometric shapes like points, lines and triangles), and typically do so at high speed (to meet the needs of data interaction, see §7). Further, the rendering system faces the daunting challenge of successfully engaging with the visual system of the human observer. It is not simple enough to draw millions of primitives, no matter how fast, because overwhelming the user with data does not help in the efficient transmission of meaningful information. Thus effective rendering strategies must work closely with the data processing pipeline to extract and filter relevant information.

Upcoming Challenges:

Some of today's most rapidly evolving computing technologies address the challenge of exchanging data with the user, in other words human-computer interaction. Such interaction may involve a variety of delivery considerations ranging from the hardware platform (supercomputer, desktop, mobile phone or tablet); rendering technology (GPU or CPU, and the supporting software libraries such as OpenGL and CUDA); and the means of delivery (web browser, mobile or desktop application). These myriad choices, combined with rapidly changing technologies, place a significant burden on visualization systems. Complex, large visualization applications may require extensive rework to support a new rendering technology in a consistent manner, only to see the technology become obsolete shortly thereafter. Smaller, more agile applications can adapt quickly, but fail to provide an integrated environment in which to perform sophisticated visual analysis.

One obvious approach to addressing this challenge is to architect rendering subsystems as independent modules and then build on these interchangeable modules. While this works well for many visualization applications, often there are special capabilities found in one rendering architecture that do not easily translate into others. Volume rendering is

a classic example–use of complex GPU capabilities can produce extreme speeds, capabilities that a device like a tablet may not be able to provide. Another complication is the delivery of data; large data may require separation of the data (server) from the rendering client, including clients such as WebGL available on many web browsers. However, such separation requires careful coordination and control of the data being passed between the server and client. Otherwise, excessive latency or even program failure may result as data overloads the rendering system.

Evolutionary/Revolutionary: 80% / 20%

Most of the work to support the variety of rendering options involves developing clever software architectures and adaptors to leverage emerging technologies. In some cases, applications can be designed to use different strategies depending on the particular delivery platform. Another approach is to abstract and simplify the number of supported rendering capabilities; using this reduced palette, which can be designed for portability and efficiency, can produce more agile visualization systems.

7 Interaction

Interaction deals with how users direct visualization software and how they gather results. A common usage model has a user employing a mouse to click buttons in a graphical user interface to direct visualization software and viewing pictures on their desktop to gather the results. Here we explore the visualization software ramifications of improved interactions. Two topics dominate: (1) how does Big Data change the nature of interaction? And (2) what visualization software improvements are necessary to further realize the potential of virtual reality?

The importance of interaction varies over the type of analysis being performed. For confirmative analysis, an animated visualization is most often adequate to confirm or withdraw a hypothesis about the character of a simulated phenomenon, and interaction is not critical. For exploratory analysis, however, interaction is very important. Exploratory analyses are characterized by a trial-and-error process, where domain experts vary visualization methods, visualization parameters, and views in an interactive session. Visualization tasks that are performed by the user with high frequency must be answered as fast as possible, if not in real-time.

Upcoming Challenges

The interactivity requirement creates special challenges when it comes to Big Data. Where current approaches typically use parallelization to maximize the overall speedup of visualization algorithms, in explorative tools scheduling strategies must be totally different in a way that they take into account both latency-issues and the user's interaction behavior. A suitable architecture would adapt between utilizing HPC resources and local resources, similar to that described in §6. With HPC resources, data can be accessed at its full resolution (but with rather high latency), while on local resources, computation and rendering can occur on reduced data only (but with extremely low latency). Although §4 discusses the need for flexible application architecture for visualization software, it merits its own mention here: short of highly optimized hardware, the requirements that systems treat Big Data and provide an interactive experience are incompatible. Visualization software must be flexible enough to adaptively switch between presenting results from multiple resources based on context. There are many systems research directions that will need to be incorporated into visualization software to accomplish this goal. What are the scheduling strategies for data loading? How can they predict user interaction inquiries? How can navigation in time be optimized?

While interactivity definitely becomes more and more important, it has not been proved yet whether Virtual Reality (VR) will make its way as a widely accepted feature in the visual analysis of scientific data. However, due to the recent availability of large stereoscopic monitors, low-cost tracking systems like the Microsoft Kinect™, and alternative input methods found on tablets and smartphones, the next generation of scientists will become increasingly comfortable with VR devices. Further, several studies give a strong hint that interactive scientific visualization cannot only significantly profit from stereoscopic, but in particular from user-centered projections with wide field of regard, as they are realized in large Virtual Reality displays like PowerWalls or CAVEs (see e.g., [LSSB12]). While all of these devices have been incorporated into visualization programs previously, supporting them may become a requirement for future visualization libraries. Although formalized support can fit well within these libraries, additional abstractions are necessary for input devices, interpreting movements from input devices, and display across a variety of output devices.

Evolutionary / Revolutionary: 50% / 50%

While extensions for ever-more-prevalent interaction devices is on the evolutionary path for visualization software, the conflicting requirements of interactivity and Big Data demand new software architectures that distribute data over local and remote resources. Beyond massive parallelism, intelligent scheduling and data reduction techniques will have to be developed and incorporated into the core of any visualization software framework in order to achieve full interactivity.

8 Conclusion

This article describes challenges facing visualization software. Big Data is a common theme throughout, and it will be one of the major drivers behind many upcoming changes in visualization software. While we often know the form solutions must take, we still do not know the details behind the form; the topics discussed truly are research challenges for the field of visualization software.

Over the last twenty-five years, design patterns in visualization software have emerged that encourage modularization and isolation of complexity. The challenge going forward will be to establish new designs that continue this trend while supporting requirements in parallelization, processor architecture, application architecture and data management, data models, rendering, and interactions. A second challenge is how to adapt existing community efforts, which represent millions of lines of code and thousands of developer years, to deal with upcoming challenges. While some of these challenges may fit naturally and require modest resources (such as adding support for a Kinect™), other challenges seem to require near total re-writes (such as dealing with many-core architectures). Clearly, given the myriad open questions facing visualization software and the large investment necessary to make it a success, the community is well served to coordinate their efforts going forward.

Visualization libraries started over twenty-five years ago are still in active use today. As we re-orient our software to deal with upcoming challenges, we must ask ourselves how new efforts can last for the next twenty-five years. Successfully dealing with diverse processor architectures, distributed systems, diverse data sources, massive parallelism, input and output devices, and interactivity will go a long way towards future-proofing our efforts, but the lessons learned from the past – abstraction, interfaces, and design patterns – will undoubtedly help lead the way.

References

[AGL05] AHRENS J., GEVECI B., LAW C.: Visualization in the paraview framework. In *The Visualization Handbook* (2005), Hansen C., Johnson C., (Eds.), pp. 162–170. 2

[AT95] ABRAM G., TREINISH L. A.: *An extended data-flow architecture for data analysis and visualization*. Research report RC 20001 (88338), IBM T. J. Watson Research Center, Yorktown Heights, NY, USA, Feb. 1995. 2

[CBW*11] CHILDS H., BRUGGER E., WHITLOCK B., MEREDITH J., AHERN S., BONNELL K., MILLER M., WEBER G. H., HARRISON C., PUGMIRE D., FOGAL T., GARTH C., SANDERSON A., BETHEL E. W., DURANT M., CAMP D., FAVRE J. M., RÜBEL O., NAVRÁTIL P., WHEELER M., SELBY P., VIVODTZEV F.: 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. 2

[CPA*10] CHILDS H., PUGMIRE D., AHERN S., WHITLOCK B., HOWISON M., PRABHAT, WEBER G., BETHEL E. W.: Extreme Scaling of Production Visualization Software on Diverse Architectures. *IEEE Computer Graphics and Applications 30*, 3 (May/June 2010), 22–31. 2

[HBC12] HOWISON M., BETHEL E. W., CHILDS H.: Hybrid Parallelism for Volume Rendering on Large-, Multi-, and Many-Core Systems. *IEEE Transactions on Visualization and Computer Graphics 18*, 1 (2012), 17–29. 2

[LSA12] LO L.-T., SEWELL C., AHRENS J.: PISTON: A portable cross-platform framework for data-parallel visualization operators. Eurographics Symposium on Parallel Graphics and Visualization, pp. 11–20. 4

[LSSB12] LAHA B., SENSHARMA K., SCHIFFBAUER J. D., BOWMAN D. A.: Effects of immersion on visual analysis of volume data. *IEEE Transactions on Visualization and Computer Graphics 18* (2012), 597–606. 7

[MAGM11] MORELAND K., AYACHIT U., GEVECI B., MA K.-L.: Dax Toolkit: A Proposed Framework for Data Analysis and Visualization at Extreme Scale. In *Proceedings of the IEEE Symposium on Large-Scale Data Analysis and Visualization* (October 2011), pp. 97–104. 4

[MAPS12] MEREDITH J. S., AHERN S., PUGMIRE D., SISNEROS R.: EAVL: the extreme-scale analysis and visualization library. In *Eurographics Symposium on Parallel Graphics and Visualization* (2012), The Eurographics Association, pp. 21–30. 4

[Nor11] NOREN A.: *Big Data Now: Current Perspectives from O'Reilly Radar*. O'Reilly Media, 2011. 1

[SML96] SCHROEDER W. J., MARTIN K. M., LORENSEN W. E.: The design and implementation of an object-oriented toolkit for 3d graphics and visualization. In *VIS '96: Proceedings of the 7th conference on Visualization '96* (1996), IEEE Computer Society Press, pp. 93–ff. 2

[UJK*89] UPSON C., JR. T. F., KAMINS D., LAIDLAW D. H., SCHLEGEL D., VROOM J., GURWITZ R., VAN DAM A.: The application visualization system: A computational environment for scientific visualization. *Computer Graphics and Applications 9*, 4 (July 1989), 30–42. 2