VTK-m: Accelerating the Visualization Toolkit for Massively Threaded Architectures

Kenneth Moreland ■ *Sandia National Laboratories*

Christopher Sewell ■ *Los Alamos National Laboratory*

William Usher ■ *University of Utah*

Li-ta Lo ■ *Los Alamos National Laboratory*

Jeremy Meredith and David Pugmire ■ *Oak Ridge National Laboratory*

James Kress ■ *University of Oregon*

Hendrik Schroots ■ *Intel*

Kwan-Liu Ma ■ *University of California, Davis*

Hank Childs ■ *University of Oregon*

Matthew Larsen ■ *Lawrence Livermore National Laboratory*

Chun-Ming Chen ■ *Ohio State University*

Robert Maynard and Berk Geveci ■ *Kitware*

Traditional scientific visualization software approaches do not fare well in massively threaded environments. To address the needs of the high-performance computing community, the VTK-m framework fills the gaps in functionality by bringing together the most recent research.

Although the basic architecture for high-performance computing (HPC) platforms has remained homogeneous and consistent for more than a decade, revolutionary changes are appearing on leading-edge supercomputers. Plans for future supercomputers promise even larger changes. But one troubling attribute of future HPC machines is the massive increase in concurrency required to sustain peak computation. In fact, most project billions of threads to achieve an exaflop.[1] This increase is partially accredited to requiring more cores to achieve faster aggregate computing rates and partially accredited to using additional threads per core to hide memory latency. Because of cost and power limitations, the system memory will not commensurately increase, which means algorithms will need strong scaling (that is, more parallelism per datum).

This trend toward massive threading can be seen in high-performance computing today. The current leadership-class computer at the Oak Ridge National Laboratory, Titan, requires between 70 and 500 million threads to run at peak, which is 300 times more than was required by its predecessor, JaguarPF. In contrast, the system memory grew only by a factor of 2.3.

The increasing reliance on concurrency to achieve faster execution rates invalidates the scalability of much of our scientific HPC code. New processor architectures are leading to new programming models and new algorithmic approaches. The design of new algorithms and their practical implementation are a critical extreme-scale challenge.[2,3]

To address these needs, HPC scientific visualization researchers working for the United States Department of Energy are building a new library called VTK-m that provides a framework for simplifying the design of visualization algorithms on current and future architectures. VTK-m also provides a flexible data model that can adapt to many scientific data types and operate well on multi-

 Published by the IEEE Computer Society

threaded devices. Finally, VTK-m serves as a container for algorithms designed in the framework and gives the visualization community a common point to collaborate, contribute, and leverage massively threaded algorithms.

The Challenges of Highly Threaded Visualization

The scientific visualization research community has been building scalable HPC algorithms for more than 15 years, and today there are multiple production tools that provide excellent scalability. However, our current visualization tools are based on a message-passing programming model. They expect a coarse decomposition of the data that works best when each processing element has on the order of 100,000 to 1 million data cells.

For many years, HPC visualization applications such as ParaView, VisIt, EnSight, and FieldView have supported parallel processing on distributed memory computer systems. The approach used by all these software products is a bulk synchronous parallel model, where algorithms perform the majority of their computation on independent local operations.[4]

This parallel computation model has worked well for the last 15 years. Even recent multicore processors could be leveraged reasonably efficiently as independent message-passing processes on each core, allowing these tools to scale to petascale machines.[5]

However, processors designed for HPC installations are undergoing transformative design changes. With physical limitations that prevent individual cores from executing instructions appreciably faster than their current rate, manufacturers are increasing the total computational bandwidth by adding more cores to each processor.[6] Some HPC processor designs go even further to increase the total possible execution throughput by removing latency-hiding features and incorporating vector processing. A consequence of all these features is that it is no longer sufficient to treat each core as an independent processor.

The upshot is that our parallel computing model is no longer symmetric. The relationship between two cores on the same processor differs significantly from the relationship between two nodes of a supercomputer. To address this asymmetry, a popular approach is to use a mixed-mode, or hybrid, parallel computing model that incorporates two levels of task organization. The first level comprises distributed-memory message-passing nodes in a cluster-like arrangement. Then, within each node of the distributed-memory arrangement is a processor or small set of processors capable of executing numerous threads that may require coordination.

From our point of view, the principle advantage of the mixed-mode parallel model is that we can leverage our existing software to manage the message-passing parallel units. The VTK-m framework focuses on the intranode parallelism that often requires massive threading and synchronized execution.

Thus, new HPC systems require a much higher degree of parallelism that could require threads operating on as few as one to 10 data cells. At this fine degree of parallelism, our conventional visualization breaks down in multiple different ways.

Load Imbalance

Typically, data are partitioned under the assumption that the amount of work per datum is uniform. However, this is not true for all visualization algorithms, many of which generate data conditionally based on the input values. With only a few exceptions, current parallel visualization functions completely ignore this load imbalance, which is considered tolerable when amortized over larger partitions.

When the data gets decomposed to the cell level, this amortization no longer occurs, which results in a much more severe load imbalance. Finely threaded visualization algorithms need to be cognizant of potential load imbalance and schedule work accordingly.

Dynamic Memory Allocation

When the amount of data a visualization algorithm generates is dependent on the values of the input data, the output data's size and structure is not known at the execution outset. In such a case, the algorithm must dynamically allocate memory as data are generated.

Because our conventional parallel visualization algorithms operate on coarse partitions in distributed memory spaces, processing elements can dynamically allocate memory completely independent from one another. In contrast, dynamic memory allocation from many threads within a shared memory environment requires explicit synchronization that inhibits parallel execution.

Topological Connections

Scientific visualization algorithms are dominated by operations on topological connections in meshes. Care must be taken when defining these connections across boundaries of data assigned to different processing elements. Mutual data being read must be consistent, and mutual data being written must be coordinated.

Predecessors of VTK-m

Although the VTK-m software project itself started little more than a year ago, the software originated as an aggregation of three predecessor products: PISTON, Dax, and EAVL. The US Department of Energy (DoE) high-performance computing (HPC) community predicted early on that leadership class facilities would be transitioning to heavily threaded processors and that we would need a significant change to our visualization algorithms and software.[1] Consequently, researchers at the DoE national laboratories began considering the challenges of visualization on accelerator processors and created three separate toolkits, each focusing on a specific aspect of the problem.

The first toolkit, PISTON,[2] considers the design of portable multithreaded visualization algorithms. Built on top of the Thrust library,[3] algorithms in PISTON comprise a sequence of general parallel operations. Originally designed for CUDA, Thrust has a flexible device back end that now supports multicore CPU architectures. PISTON has demonstrated scalability across multiple devices.

The second toolkit, Dax,[4] considers a top-down approach to a multithreaded visualization framework. Recognizing that most visualization algorithms iteratively apply an operation to each element in a mesh, the Dax framework lets developers concentrate on designing these per-element operations while the framework automatically builds the appropriate parallel scheduling.

The third toolkit, EAVL,[5] considers the data model used in modern visualization. EAVL updates the visualization data model to handle modern simulation codes using flexible mesh structures that are also easily accessible in multithreaded environments.

Each originally focused on a different part of the extreme-scale visualization problem, but the software packages did not integrate well. Recognizing the prospect of substantial duplication of effort, the developers of these software projects came together to work under a unified software product: VTK-m. Although VTK-m was born from a new code base, the PISTON, Dax, and EAVL developers contributed and evolved their respective technologies. The development of its predecessors has been phased out, and VTK-m is now a unified, well-integrated product of these three predecessors with continuing evolving capabilities.

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," Dept. of Energy Office of Advanced Scientific Computing Research, Feb. 2011; http://science.energy.gov/~/media/ascr/pdf/program-documents/docs/Exascale-ASCR-Analysis.pdf.
2. L. Lo, C. Sewell, and J. Ahrens, "PISTON: A Portable Cross-Platform Framework for Data-Parallel Visualization Operators," *Proc. Eurographics Symp. Parallel Graphics and Visualization* (EGPGV), 2012, pp. 11–20.
3. N. Bell and J. Hoberock, "A Productivity-Oriented Library for CUDA," *GPU Computing Gems, Jade Edition*, W.W. Hwu, ed., Morgan Kaufmann, 2011, pp. 359–371.
4. K. Moreland et al., "Flexible Analysis Software for Emerging Architectures," *Proc. SC'12 Companion: High Performance Computing, Networking Storage and Analysis*, 2012, pp. 821–826.
5. J.S. Meredith et al., "EAVL: The Extreme-Scale Analysis and Visualization Library," *Proc. Eurographics Symp. Parallel Graphics and Visualization*, 2012, pp. 21–30.

A conventional parallel visualization algorithm typically manages topological connections across processes by replicating data on the boundaries. Because the partitions are coarse, there is a limited amount of replication. However, this replication cannot be sustained as the data decomposition approaches single cells. Instead, threads must share connected data, and generated data may require further processing to identify coincident topology.

SIMD Execution

In addition to increasing execution bandwidth through multiple processing cores, modern HPC processors also enable vector processing, which allows the same instruction to be executed simultaneously on multiple data values. Whereas conventional parallel visualization algorithms allow for completely independent execution, efficient vector processing requires at least some parallel execution to run in single instruction, multiple data (SIMD) mode. New algorithms should minimize code execution divergence.

Visualization with Data Parallel Primitives

Data parallel primitives provide an abstraction layer between the low-level hardware architecture and the high-level code, which both increases the portability of VTK-m and simplifies the implementation of algorithms. Guy Blelloch originally proposed a scan-vector model for data parallel computing and outlined a variety of algorithms in data structures, computational geometry, graphs, and numerical analysis.[7] All of VTK-m's predecessors (PISTON, Dax, and EAVL) leverage data parallel primitives in some way (see the "Predecessors of VTK-m" sidebar for more details). VTK-m provides an abstract device model inherited from Dax and expanded to fulfill the needs of the algorithms inherited from PISTON. VTK-m makes use of the following data parallel primitives.

- *Map* applies the same unary operator to each element of an array in parallel. This is useful for element-wise field expressions or can be leveraged as a general scheduling capability.
- *Reduce* computes a single value (such as the sum or product) from all the elements of an array, which is useful whenever aggregate values are needed.
- *Scan* is similar to a reduction but stores intermediate results (that is, a running total) at each element of the array. Scans are particularly useful for finding array indices.
- *Sort* is a versatile reordering operation that is particularly useful for identifying all groups of common values in an array.
- *Search* can simultaneously find multiple values in an array. This can be used to find a reverse mapping of values in one input array to their locations in a second sorted input array.

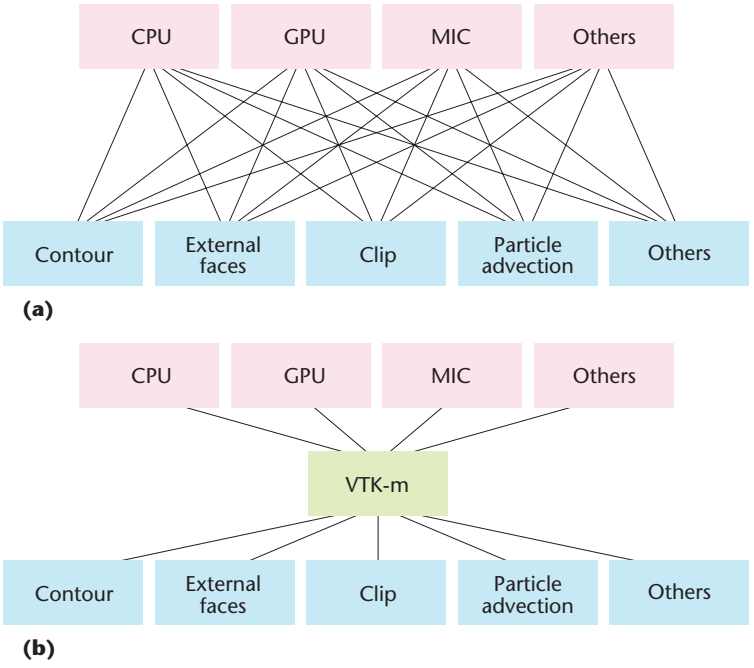

Figure 1. VTK-m's abstract device model. (a) Implementing all of our visualization algorithms separately for every architecture we need to support leads to an unmanageable amount of software. (b) By using VTK-m's abstract device model based on data parallel primitives, we can reduce the amount of software to implement to a feasible level.

Algorithms that employ hierarchical parallelism may make use of segmented versions of data parallel primitives. For example, when constructing KD-trees or other types of graphs, the graph's structure may be encoded using a segment vector, where each segment represents a node. Operations such as sorts and scans can then be performed independently within each node but still all in parallel.

Although research has found effective ways to utilize these data parallel primitives for many visualization problems, the application of these primitives is not always obvious. Fortunately, many visualization algorithms follow similar patterns of operation,[8] and these patterns are embodied by objects called worklets in VTK-m, a concept inherited from Dax.

Worklets extend the functionality of the basic data parallel primitives by providing common features. For example, several flavors of worklets can trace topological connections in map operations. VTK-m handles the indexing of the topology structure internally, which makes it easier and safer to write worklets than to directly operate on data structures in parallel. Other worklet types can manage algorithms with sparse output or find coincident topology, further reducing the burden on the programmer.

Performance Portability with Data Parallel Primitives

Unlike many "in house" HPC software applications that can be written for a specific HPC installation, the software our team develops is used across all the US DoE supercomputer facilities and many others around the world. This means our software must perform well on a variety of hardware types and configurations. Furthermore, the critical code in a visualization system is not limited to a small number of iterative loops. Our software contains numerous algorithm implementations to address visualization needs across many scientific disciplines. For example, the Visualization Toolkit, on which ParaView and VisIt are based, contains more than 400 distinct filters.

Upgrading this functionality is a daunting challenge. Attempting to create multiple algorithm implementations targeted to each known platform (and possibly currently unknown platforms in the future) creates a combinatorial explosion of work. This approach would require significant developer investment, likely exceeding worldwide funding available for this task.

Instead, VTK-m uses data parallel primitives to achieve performance portability. VTK-m defines an abstract device model constructed from the aforementioned data parallel primitive operations (such as map, scan, sort, and reduce) that run in parallel on the device. The VTK-m team is rethinking visualization algorithms as a sequence of these parallel primitive operations. When implemented as such, the entirety of algorithms in VTK-m can be ported to a new device by providing only the parallel primitives for that device. Algorithms need only be written once to VTK-m's abstract device model, which dramatically reduces the implementation work for various architectures (see Figure 1).

The indirection of using data parallel primitives to implement algorithms can limit the amount

of optimization that can be done, which raises the question of how much performance portability costs. Evidence collected from recent research using data parallel primitives[9,10] including work presented here suggests a fairly low overhead. The benefits of simplified implementation and performance portability outweigh any overheads introduced by the approach.

Advanced Data Models

VTK-m is intended to be used in many scientific disciplines and with data of many different arrangements and structures. It is important that the data model in VTK-m be flexible enough to capture the different representations of data while still providing clear semantics for the elements in the data. Furthermore, the data representation must be space efficient and accessible on the different processor types we use (that is, work on both CPUs and GPUs). To achieve these goals, VTK-m provides a data model inherited from its predecessor EAVL (see the "Predecessors of VTK-m" sidebar) with some modifications and extensions.

The utility of the data models is further enhanced by integrating them with the aforementioned data parallel primitives, abstract device adapters, and worklet mechanisms. Datasets in VTK-m seamlessly move between devices. Algorithms built within VTK-m use abstract concepts that can be applied to datasets of many different structures without having to be reimplemented.

Datasets

Traditional dataset models often choose a set of rigid characteristics for a dataset and then label that collection of characteristics as a specific type of mesh. For example, a uniform dataset has regular axis-aligned coordinates and a logical [*i*, *j*, *k*] cell arrangement. An unstructured dataset has fully explicit coordinates (an [*x*, *y*, *z*] value separately defined for each point) with fully explicit cell connectivity defined by arrays of indices. A curvilinear grid is a mix with logical cell arrangement but explicit coordinates.

However, this selection of grid types is somewhat arbitrary. It is easy to generate unsupported meshes. For example, consider a tetrahedralization of a uniform grid. This would be most efficiently represented with regular coordinates and explicit cells, but most data models require a fully explicit unstructured grid.

VTK-m removes this restriction entirely by defining a dataset as a mere container for arbitrary collections of coordinates, cell sets, and fields. Although this allows VTK-m to support more mesh types, the algorithm designer does not have to manage this complexity. At the worklet level, cell shape and field values are decoupled from the mesh representation, meaning the algorithm need only be implemented once for all mesh types.

The ability to correctly represent mesh structures is important for computational efficiency and accuracy. It is also critical for in situ visualization and analysis tasks because any mismatch between the representational capabilities of the analysis code relative to the targeted simulation code will result in expensive (both computationally and in terms of memory footprint) copies and transformations of data arrays.

Coordinate Systems

Because it allows an arbitrary collection of coordinates, VTK-m supports datasets with no coordinates (such as graph data with no physical location for the vertices) or datasets with coordinate systems of spatial dimension other than three. VTK-m also supports more than one coordinate system, which recognizes that a mesh might be interpreted in more than one spatial mapping independent of the underlying mesh structure. For example, geospatial data could simultaneously have coordinate systems defined by 3D position, latitude-longitude coordinates, and any number of 2D projections.

Cell Sets

A common problem in traditional dataset models is the inability to properly support more than one type of topological element in the same mesh, such as volumetric elements with boundary interface polygons or point elements with connecting line segments. Although it can often be supported by merging these disparate topological element types into a single mesh, the lack of separation requires that fields contain dummy values for the element types for which they are inapplicable, an error-prone and inefficient practice.

VTK-m solves this problem by allowing multiple cell sets to be contained in the same mesh. The unifying characteristic of belonging to the same mesh (as opposed to two different meshes) is that they both refer to the same set of points using the same set of indices, which allows efficient mapping between these two cell sets.

Use Cases

There has been significant interest within the scientific visualization community to experiment with and adopt the VTK-m framework. This section contains several examples of VTK-m being used in a variety different projects, including volume render-

ing, ray tracing, geometric operations, and workflows for large-scale scientific applications.

Ray Tracing

Matthew Larsen and his colleagues implemented a ray tracer using data parallel primitives in an effort to provide a many-core capable infrastructure that can perform rendering on any architecture.[10] Ray tracing is a compelling algorithm for exploring the boundaries of data parallel primitives because it is computationally intensive and has unstructured memory accesses. Furthermore, the HPC community has a growing interest in using ray tracing rather than the traditional rasterization-based rendering because the computational complexity of ray tracing is much better with respect to the size of the geometry (which has been steadily growing in the HPC setting).

To verify the efficiency of the VTK-m framework, we implemented Larsen's ray-tracing algorithm in VTK-m and replicated many of the experimental runs in the Larsen paper.[10] Figure 2 shows example output from the ray tracer, and Figure 3 summarizes the results of our experiments. The datasets are the same ones used in the Larsen study, as are the timings for the Optix Prime, Embree, and EAVL algorithms. The GPU runs come from a Tesla K40, and the CPU runs come from dual Intel Xeon E5's, which are equivalent to the GPU2 and CPU2 configurations, respectively, in the Larsen paper. Likewise, all images are rendered with 1,080p resolution (1920 × 1080), as the Larsen paper reported.

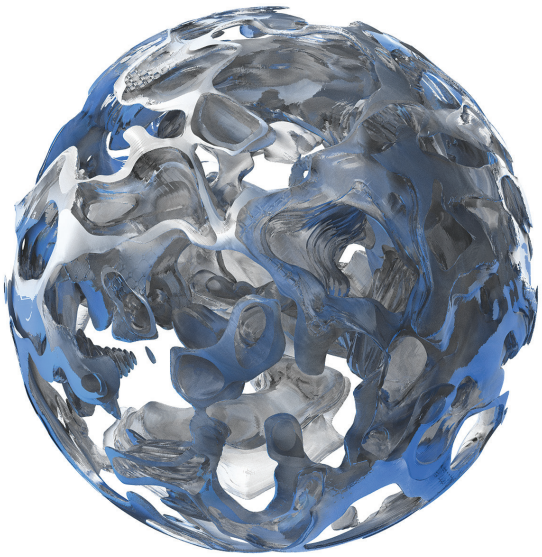

Figure 2. Example output from VTK-m ray-tracing algorithm. The data in this rendering represent seismic wave propagation through the Earth.

Our experiments verify the conclusions drawn by the Larsen study in that our architecture-agnostic data parallel approach is competitive with highly optimized architecture-specific approaches (within a factor of two). Also, the VTK-m implementation is at least as fast as, and sometimes faster than, Larsen's previous implementation written using

Dataset	Algorithm	Millions of rays per second
LT_350K	OptiX Prime	357.6
	EAVL	150.8
	VTK-m	164.5
LT_372K	OptiX Prime	322.4
	EAVL	124.7
	VTK-m	140.8
RM_350K	OptiX Prime	436.5
	EAVL	197.5
	VTK-m	200.8
RM_650K	OptiX Prime	420.4
	EAVL	172.9
	VTK-m	166.0
RM_970K	OptiX Prime	347.1
	EAVL	152.8
	VTK-m	163.5
RM_1.7M	OptiX Prime	266.8
	EAVL	136.6
	VTK-m	148.8
RM_3.2M	OptiX Prime	264.5
	EAVL	124.8
	VTK-m	134.5
Seismic	OptiX Prime	267.8
	EAVL	106.3
	VTK-m	119.4

(a)

Dataset	Algorithm	Millions of rays per second
LT_350K	Embree	51.9
	EAVL	27.7
	VTK-m	38.5
LT_372K	Embree	56.5
	EAVL	26.1
	VTK-m	36.0
RM_350K	Embree	64.8
	EAVL	33.3
	VTK-m	47.8
RM_650K	Embree	65.9
	EAVL	35.6
	VTK-m	49.1
RM_970K	Embree	59.1
	EAVL	29.3
	VTK-m	41.0
RM_1.7M	Embree	52.4
	EAVL	27.0
	VTK-m	37.8
RM_3.2M	Embree	48.4
	EAVL	28.3
	VTK-m	33.9
Seismic	Embree	43.2
	EAVL	25.2
	VTK-m	34.5

(b)

Figure 3. The speed (measured in millions of rays cast per second) to perform ray tracing using different algorithms on different data sets. (a) GPU times and (b) CPU times. The datasets are the same ones used in the Larsen study,[10] as are the timings for the Optix Prime, Embree, and EAVL algorithms.

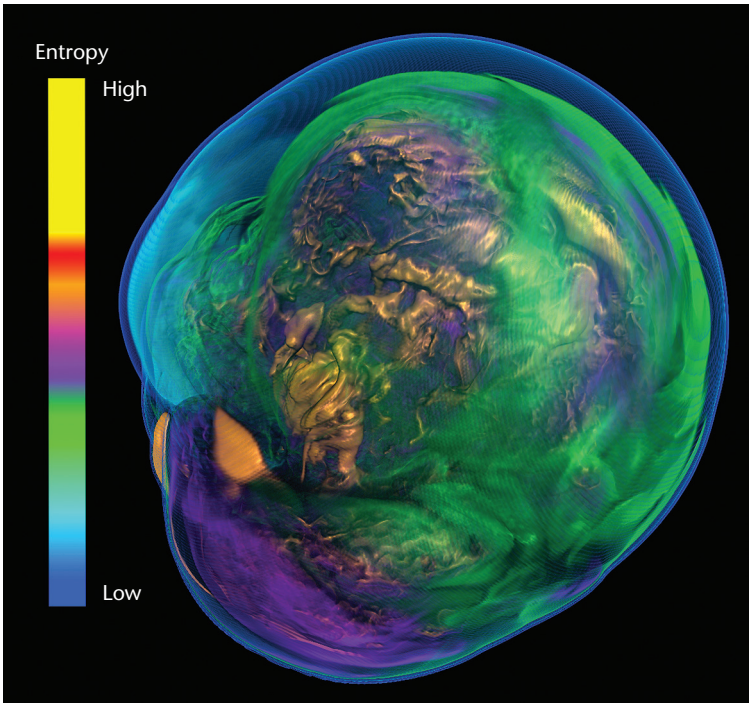

Figure 4. Example output of the VTK-m volume renderer. The rendered image is from supernova simulation data using a direct volume renderer implemented in VTK-m.

EAVL, which suggests that the integration of data models with the execution model, algorithms, and other features of VTK-m does not add overhead.

Direct Volume Rendering

Direct volume rendering, in which we render a volume in space with variable color and opacity, is a common but computationally intensive task in scientific visualization. Thus, direct volume rendering is a good use case to demonstrate VTK-m's capabilities. We implemented a simple ray-casting direct volume renderer within the VTK-m framework. Figure 4 shows an example output of this volume renderer.

For testing, we used the output of a supernova simulation dataset, consisting of a 432^3 volume of voxels and rendered to a 1,024 × 1,024 window. Because performance is directly related to the number of rays that intersect the volume, and ultimately generate a color, the camera is positioned so that the volume occupies a large portion of the viewing window. The rendering time is averaged over the rendering of 16 frames.

We tested the same VTK-m volume rendering on three different devices: a dual-processor Intel Xeon IvyBridge E5-2670 v2 (labeled CPU), an NVIDIA Titan X (labeled GPU), and an Intel Xeon Phi 5110p coprocessor (labeled MIC). For comparison purposes, we also rendered the volume using VTK (with the vtkFixedPointVolumeRayCastMapper class). This VTK implementation is multithreaded but designed strictly for the CPU so it was only run on that architecture. We also compared our VTK-m volume rendering algorithm with a similar one implemented in Dax. Figure 5 summarizes the results of these experiments. All implementations were multithreaded and run on all cores available.

The comparison between our VTK-m volume rendering and the existing VTK volume rendering is not a direct comparison of the performances of the two respective libraries because there are significant differences between the two algorithm implementations. In particular, the VTK-m volume rendering is much simpler than the VTK volume rendering, which has optimizations such as empty space skipping, which is not yet implemented in our algorithm. Nevertheless, our simpler algorithm performs comparatively well and is also portable across multiple architectures.

In contrast, the volume rendering implementations in VTK-m and Dax are similar. We can clearly see that the integration of the Dax technologies with those of the other VTK-m predecessors has not resulted in a larger overhead. On the contrary, the VTK-m framework operated more efficiently.

Isosurface

Isosurfacing is one of the most valuable and widely used algorithms in scientific visualization. Our isosurface algorithm is adapted from PISTON. The new implementation integrates the algorithm with VTK-m's data model and worklet execution mechanism.

We demonstrate the isosurface algorithm using a 432^3 supernova dataset similar to the one used

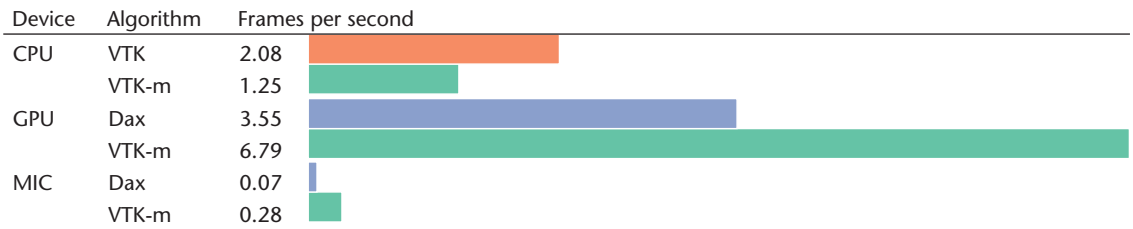

Figure 5. Average rendering rate for a 1,024 × 1,024 volume rendering image. All implementations were multithreaded and run on all cores available.

in the direct volume rendering experiments. Figure 6 shows an example contour generated. The CPU run times were measured with two Xeon E5-2698 processors, each with 16 cores and two times hyperthreading running at 2.3 GHz. The CUDA runs were measured on a Tesla K40. Figure 7 gives a summary of the time.

We compared the VTK-m algorithm with the similar algorithms in PISTON and a serial algorithm in VTK that can be run in parallel with MPI. To make the comparison as authentic as possible, we used VTK's Marching Cubes implementation of contouring (vtkMarchingCubes) without point merging. The MPI experiments were run on a single node.

In serial, our VTK-m Marching Cubes implementation ran at about half the speed of the VTK implementation. This is likely due to duplicate computation during the two-pass algorithm in VTK-m. However, the VTK-m algorithm exceeded the VTK algorithm's performance in parallel, almost doubling the speed, which we attribute to there being load imbalance when using MPI to parallelize the old VTK version. The exact same VTK-m implementation also had good performance on the CUDA device on which it ran faster than any of the parallel CPU implementations. The performance of the VTK-m implementation is comparable to the previous generation algorithm implemented in PISTON. The VTK-m implementation is slightly slower, which we attribute to the PISTON implementation consolidating some index computations.

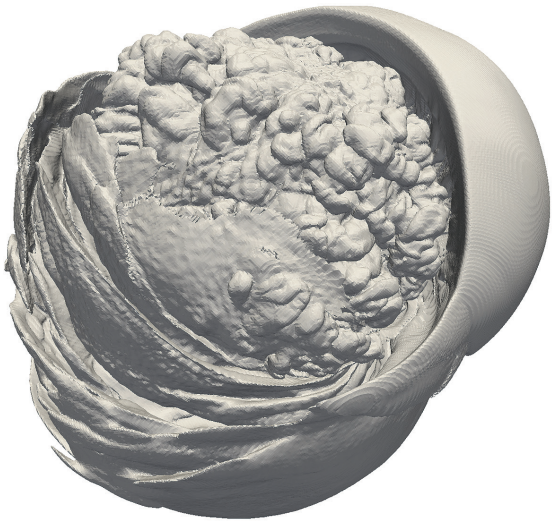

Figure 6. Example contour generated by VTK-m. The 432^3 supernova dataset is similar to the one used in the direct volume rendering experiments.

Surface Simplification

The simplification of polygonal meshes is a critical component of visualization applications that wish to provide rendering levels of detail. ParaView uses a point-clustering algorithm that combines mesh points using a regular grid of bins,[11] as Figure 8 demonstrates. We implemented this algorithm in VTK-m, which performs the clustering by identifying the bins in a map operation and then combining points in the same bin by sorting based on bin identifier.

We demonstrated this algorithm using the Lucy data from the Stanford 3D scanning repository (http://graphics.stanford.edu/data/3Dscanrep/). The experiments clustered using 512^3, $1{,}024^3$, and $2{,}048^3$ bins, which reduced the output by 97, 91, and 67 percent, respectively. For comparison purposes, we can demonstrate the VTK-m algorithm in serial and the serial VTK algorithm (except for the $2{,}048^3$ grid, for which the VTK filter ran out of memory). The CPU runtimes were measured with two Intel Xeon E5-2699 processors, each with 16 cores and two times hyperthreading running at 2.3 GHz with 64 Gbytes of memory. The CUDA runs were measured on a Tesla K40. Figure 9 gives a summary of the time.

Even when run in serial, the VTK-m implementation of surface simplification takes less time than the VTK implementation. This difference is mostly because the VTK-m algorithm uses a simpler but effective vertex positioning that requires

Algorithm	Device	Time (sec)
vtkMarchingCubes	Serial	11.917
vtkMarchingCubes	32 MPI ranks	1.352
vtkMarchingCubes	64 MPI ranks	1.922
PISTON	Serial	19.895
PISTON	CUDA	0.514
PISTON	TBB	0.955
VTK-m	Serial	20.784
VTK-m	CUDA	0.560
VTK-m	TBB	1.161

Figure 7. Time (in seconds) to run different isosurface implementations on various devices with different output sizes. The VTK-m algorithm exceeded the VTK algorithm's performance in parallel, almost doubling the speed.

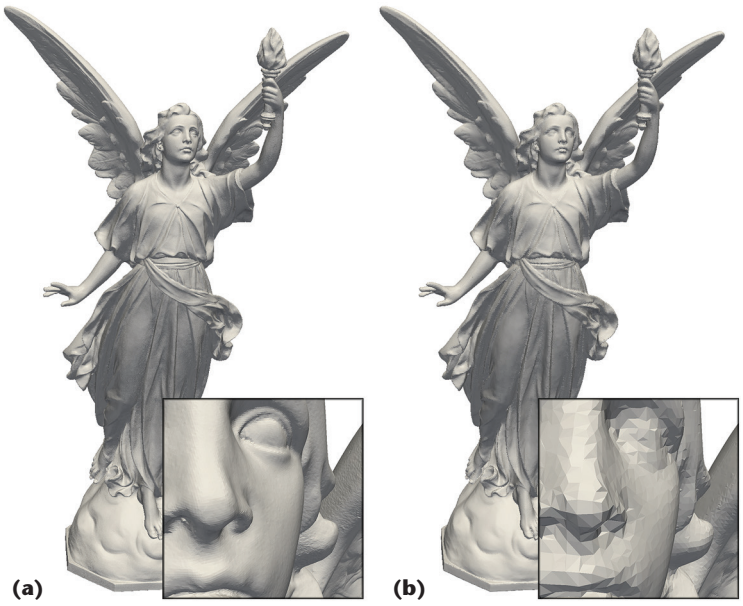

Figure 8. Example of mesh simplification. (a) Original mesh and (b) a mesh simplified on a 512^3 grid of bins.

less computation and memory. Running the VTK-m algorithm in parallel further reduces the running time. We did observe a limit in the speedup for this algorithm, particularly for Intel Threading Building Blocks (TBB). This appears to be a limit of the sort algorithm provided by TBB, which dominates the runtime.

Integration with HPC Applications

Visualization has long been a critical tool used by scientific applications for a variety of tasks, including data exploration, debugging, and results validation and presentation. The size and complexity of the data generated by these application codes has pushed more and more visualization onto larger and more massively threaded and heterogeneous resources, including the supercomputer running the application using an in situ visualization paradigm.

VTK-m is designed to assist these applications by running on both dedicated visualization resources and with the application on the supercomputer using an in situ paradigm. We have been exploring the integration of VTK-m with XGC1, a highly scalable physics code used to study plasmas in fusion tokamak devices.[12] Recent work has explored using lightweight plug-ins to perform visualization both of the particles and the field variables in XGC1 using ADIOS and DataSpaces as a lightly coupled in situ framework.[13]

XGC1 production runs produce tremendous amounts of data, particularly particle data. Thus, XGC1 scientists are interested in visualization and analysis of these particles in numerous ways that include both the entire set of particles and various subsets of particles based on particular properties. Furthermore, these analyses might take place in different parts of the data workflow, from simulation nodes, loosely coupled nodes, or disk. Depending on this workflow, the visualization and analysis code must perform well in a variety of different locations with different computer hardware. VTK-m's performance portability offers this flexibility while maintaining low overhead. Also, because of the size of this particle data, efficient representations on the simulation nodes and coupling nodes are paramount. VTK-m's data model enables an efficient representation while taking full advantage of the data-parallel framework to process large numbers of particles. The initial work in visualization plug-ins was done using EAVL, but it is now being migrated over to the VTK-m framework.

Figure 10 shows visualizations of particles from a recent XGC1 run. One particular interest to the scientists is understanding the interactions between the plasma particles and the tokamak containment vessel. The visualization workflow consists of identifying particles that begin in the plasma's central core at the start of the simulation and then migrate over time outside the central core and collide with the tokamak wall. The visualization plug-ins we are developing for this application use ADIOS for the I/O, both file based and in situ. The in situ integration applies the VTK-m data model to the simulation particles and the tokamak wall, which lets VTK-m operate on these data without copying them. VTK-m worklets are used to first identify the particles inside and outside of the plasma core and then to flag particles that intersect with the tokamak wall.

Algorithm	Device	512^3 (sec)	$1{,}024^3$ (sec)	$2{,}048^3$ (sec)
VTK	Serial	3.65	11.40	
VTK-m	Serial	2.73	2.93	5.22
VTK-m	TBB (36 threads)	0.36	0.45	0.72
VTK-m	TBB (72 threads)	0.41	0.49	0.74
VTK-m	CUDA	0.18	0.19	0.20

Figure 9. Time (in seconds) to run the surface simplification algorithm on various devices with different output sizes. The CUDA time does not include data transfers to or from the device.

We plan to continue developing these and other visualization plug-ins for XGC1 as well as other codes running on HPC resources. We are particularly interested in furthering our exploration of these plug-ins executing in loosely coupled in situ environments like ADIOS.

Many of the scientific visualization applications in use today have roots in software that began development more than 20 years ago. Enhancements have been added over the years to address the needs of the HPC community, but these traditional approaches do not fare well in massively threaded environments. VTK-m fills this gap in functionality by bringing together the most recent research.

However, VTK-m is not intended to replace VTK or any of the other existing visualization tools, such as ParaView and VisIt. Rather, our plan is to integrate VTK-m with these large, existing software applications to enhance their performance. VTK-m exists as a separate entity so that it may be structured into the best possible framework for multithreaded development environments.

Many visualization researchers and developers are adopting VTK-m for their scientific visualization needs. We expect VTK-m to be a major resource for visualization software for many years to come.

Acknowledgments

We thank Kewei Lu from the Ohio State University for valuable contributions to the VTK-m isosurface implementation. This material is based on work supported by the US Department of Energy (DoE), Office of Science, Office of Advanced Scientific Computing Research, under award numbers 14-017566 and 12-015215. Sandia National Laboratories is a multiprogram laboratory managed and operated by Sandia, a wholly owned subsidiary of Lockheed Martin, for the US DoE's National Nuclear Security Administration under contract DE-AC04-94AL85000, SAND 2016-1719 J. The original manuscript was authored by UT-Battelle LLC under contract DE-AC05-00OR22725 with the US DoE. The US government retains and the publisher, by accepting the article for publication, acknowledges that the US government retains a nonexclusive, paid-up, irrevocable, worldwide license to publish or reproduce the published form of this manuscript, or allow others to do so, for US government purposes. The DoE will provide public access to these results of federally sponsored research in accordance with the DoE Public Access Plan (http://energy.gov/downloads/doe-public-access-plan).

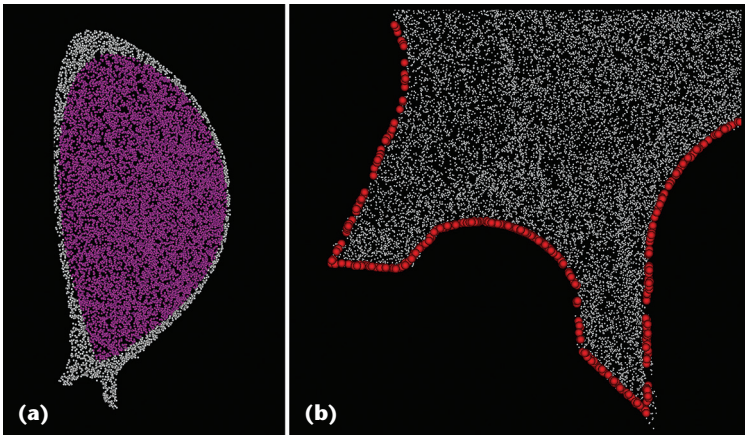

Figure 10. Particle visualization from XGC1. (a) The classification of particles inside (pink) or outside (white) of the plasma core and (b) the particles that interact with the containment wall (red).

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," Dept. of Energy Office of Advanced Scientific Computing Research, Feb. 2011; http://science.energy.gov/~/media/ascr/pdf/program-documents/docs/Exascale-ASCR-Analysis.pdf.
2. S. Ashby et al., "The Opportunities and Challenges of Exascale Computing," Summary Report of the Advanced Scientific Computing Advisory Committee (ASCAC) Subcommittee, Fall 2010; http://science.energy.gov/~/media/ascr/ascac/pdf/reports/Exascale_subcommittee_report.pdf.
3. H. Childs et al., "Research Challenges for Visualization Software," *Computer*, vol. 46, no. 5, 2013, pp. 34–42.
4. C.C. Law et al., "A Multi-threaded Streaming Pipeline Architecture for Large Structured Data Sets," *Proc. IEEE Visualization*, 1999, pp. 225–232.
5. H. Childs et al., "Extreme Scaling of Production Visualization Software on Diverse Architectures," *IEEE Computer Graphics and Applications*, vol. 30, no. 3, 2010, pp. 22–31.
6. S.H. Fuller and L.I. Millett, "Computing Performance: Game Over or Next Level?" *Computer*, vol. 44, no. 1, 2011, pp. 31–38.
7. G.E. Blelloch, Vector Models for Data-Parallel Computing, MIT Press, 1990.
8. K. Moreland et al., "A Classification of Scientific Visualization Algorithms for Massive Threading," *Proc. 8th Int'l Workshop Ultrascale Visualization* (UltraVis), 2013, article no. 2.
9. L. Lo, C. Sewell, and J. Ahrens, "PISTON: A Portable Cross-Platform Framework for Data-Parallel Visualization Operators," *Proc. Eurographics Symp. Parallel Graphics and Visualization* (EGPGV), 2012, pp. 11–20.
10. M. Larsen et al., "Ray Tracing within a Data Parallel Framework," *Proc. IEEE Pacific Visualization Symp.* (PacificVis), 2015, pp. 279–286.

11. P. Lindstrom, "Out-of-Core Simplification of Large Polygonal Models," *Proc. SIGGRAPH*, 2000, pp. 259–262.
12. C.S. Chang et al., "Compressed Ion Temperature Gradient Turbulence in Diverted Tokamak Edge," *Physics of Plasmas*, vol. 16, no. 5, 2009, p. 056108.
13. D. Pugmire et al., "Towards Scalable Visualization Plugins for Data Staging Workflows," *Proc. SC14 Workshop Big Data Analytics: Challenges and Opportunities* (BDAC-14), 2014.

Kenneth Moreland *is a principal member of the technical staff at Sandia National Laboratories. His research interests include large-scale and finely threaded scientific visualization algorithms. Moreland has a PhD in computer science from the University of New Mexico. Contact him at kmorel@sandia.gov.*

Christopher Sewell *is a staff scientist in the Computer and Computational Sciences Division at Los Alamos National Laboratory. His research interests include portable data-parallel programming models and large-scale visualization and analysis. Sewell has a PhD in computer science from Stanford University. Contact him at csewell@lanl.gov.*

William Usher *is a graduate student research assistant at the Scientific Computing and Imaging Institute at the University of Utah. His research interests include large-scale and in situ scientific visualization algorithms. Usher is currently pursuing a PhD in computer science at the University of Utah. Contact him at will@sci.utah.edu.*

Li-ta Lo *is a technical staff member at Los Alamos National Laboratory. His research interests include task and data-parallel programming for scientific visualization and simulation and big data analytics. Lo has an MS in applied mechanics from National Taiwan University. Contact him at ollie@lanl.gov.*

Jeremy Meredith *is a senior research and development staff member at Oak Ridge National Laboratory. His research interests include high-performance computing and large-scale scientific visualization. Meredith has an MS in computer science from Stanford University. Contact him at jsmeredith@ornl.gov.*

David Pugmire *is a senior staff scientist at Oak Ridge National Laboratory. His research interests include methods for scientific visualization on high-performance computers. Pugmire has a PhD in computer science from the University of Utah. Contact him at pugmire@ornl.gov.*

James Kress *is a third-year PhD student in computer science at the University of Oregon. His research interests include scientific visualization, high-performance computing, and the intersection of the two. Kress has a BS in computer science with a minor in political science from Boise State University. Contact him at jkress@cs.uoregon.edu.*

Hendrik Schroots *is a software engineer at Intel. His research interests include large-scale visualization and graphics. Schroots has an MS in visualization and graphics from the University of California, Davis. Contact him at hschroot@ucdavis.edu.*

Kwan-Liu Ma *is a professor of computer science at the University of California, Davis, where he directs VIDI Labs and the UC Davis Center for Visualization. His research interests include visualization, high-performance computing, and user interface design. Ma has a PhD in computer science from the University of Utah. He is an IEEE fellow and a recipient of the IEEE VGTC Visualization Technical Achievement Award. Contact him at ma@cs.ucdavis.edu.*

Hank Childs *is an associate professor at the University of Oregon and a staff scientist at Lawrence Berkeley National Laboratory. His research interests include large data visualization, visualization systems, and flow visualization. Childs has a PhD in computer science from the University of California, Davis. Contact him at hank@uoregon.edu.*

Matthew Larsen *is staff scientist at Lawrence Livermore National Laboratory and a PhD student at the University of Oregon. His research interests include computer graphics, scientific visualization, and HPC. Larsen received an MS in computer science from the University of Oregon. Contact him at larsen30@llnl.gov.*

Chun-Ming Chen *is a PhD candidate of the Department of Computer Science and Engineering at Ohio State University. His research interests include analysis and visualization for large flow data. Chen has an MS in computer science from the University of Southern California. Contact him at chen.1701@osu.edu.*

Robert Maynard *is an R&D engineer at Kitware. He is one of the primary developers of VTK-m with a focus on scientific visualization on heterogeneous architectures. Maynard has a BS in computer science from Laurentian University. Contact him at robert.maynard@kitware.com.*

Berk Geveci *is the senior director of scientific computing at Kitware. His research interests include large-scale data analysis and visualization within the VTK and ParaView frameworks. Geveci has a PhD in mechanical engineering from Lehigh University. Contact him at berk.geveci@kitware.com.*