The Gap between Visualization Research and Visualization Software (VisGap) (2026)
K. Blenman, C. Gillmann, M. Krone, G. Reina, T. Wischgoll (Guest Editors)

Viskores: Integrating Parallel Scientific Visualization Research into Applications

K. Moreland[1] J. Amstutz[2] T. M. Athawale[1] V. Bolea[3] M. Bolstad[4] H. Childs[5] B. Geveci[3] C. Harrison[6]
M. Larsen[7] L.-T. Lo[8] N. Marsaglia[6] M. Mathai[5] D. Pugmire[1] S. Rizzi[9] S. Tsalikis[3,10] G. H. Weber[11]

[1]Oak Ridge National Laboratory, USA [2]Nvidia Corp, USA [3]Kitware, Inc., USA [4]Sandia National Laboratories, USA [5]University of Oregon, USA
[6]Lawrence Livermore National Laboratory, USA [7]Luminary Cloud, USA [8]Los Alamos National Laboratory, USA [9]Argonne National Laboratory, USA
[10]University of North Carolina at Chapel Hill, USA [11]Lawrence Berkeley National Laboratory, USA

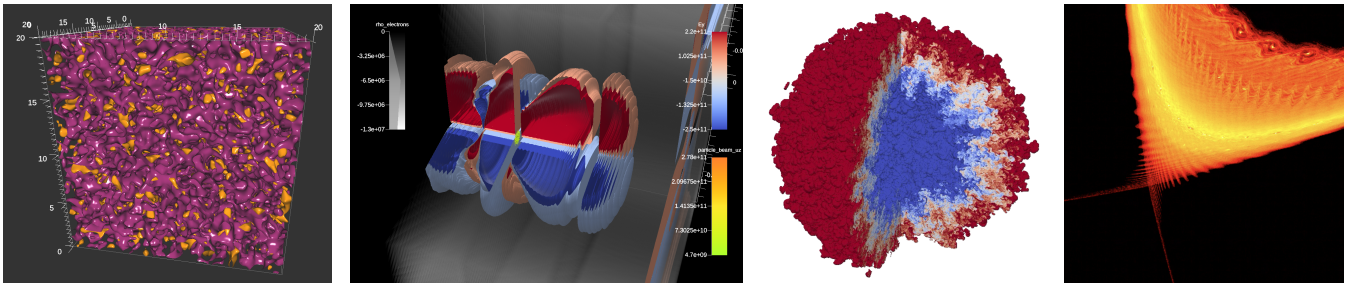

Figure 1: *Example visualizations from Viskores. From left to right: Perlin noise for a scaling study demonstrating rendering at full scale on the Frontier supercomputer, in-situ visualization of a laser-wakefield accelerator simulated by WarpX, idealized Inertial Confinement Fusion (ICF) simulation of a Rayleigh-Taylor instability with two fluids mixing in a spherical geometry, and Poincaré plot of a tokamak nuclear fusion reactor created by tracing thousands of magnetic field lines.*

Abstract

Viskores is a scientific visualization library that is the primary deployment of such algorithms to the parallel accelerated processors of modern DOE supercomputers. In this paper, we review the capabilities provided by Viskores and how these capabilities are leveraged by other software in the high-performance computing ecosystem. We discuss the Viskores data representation and pay particular attention to array management. Through this array management we describe how data is adapted between Viskores and other software along with strategies for converting dynamic, polymorphic objects to static representations better suited to GPU processing. We conclude with several examples of Viskores integrating with high-performance software that is used in production today.

CCS Concepts

• ***Human-centered computing*** → ***Visualization toolkits; Scientific visualization;***

1. Introduction

Scientific visualization has highly varied requirements in multiple areas. First, applications produce many types of data, including many types of meshes, fields, and data layouts. Second, the most meaningful techniques for visualizing data can vary across scientific domains. Third, visualization is deployed across multiple environments, i.e., stand-alone applications versus integration into larger frameworks and multiple hardware architectures.

Visualization libraries are an effective way to reduce development time for delivering diverse visualization capabilities. Rather than delivering a bespoke implementation for each use case, a library can be reused across many settings, avoiding redundant effort and reducing overall costs. The VTK library [SML04] is a canonical example of this approach, providing a single library that has been used widely across decades and domains.

As the computing landscape evolves, visualization libraries must evolve in turn. In particular, two recent shifts have placed pressure on existing designs. First, visualization capabilities are increasingly delivered in situ. In situ processing, at least in some cases, requires visualization software to be cognizant of the larger environment in-

DOI: 10.2312/visgap.20261000

cluding reduced memory footprint, adapting to existing data structures, fast execution times, moving computation to data, and more. Second, parallelism has become ubiquitous on modern compute architectures, with architectures like GPUs having tens of thousands of cores. As data sizes grow, visualization libraries must leverage this parallelism to achieve sufficient performance. Further, in situ workflows often require visualization to execute where the simulation runs, frequently on parallel hardware and under tight performance constraints.

Viskores fills this recent gap in the evolving landscape of high-performance computing. Born with massively-threaded execution and memory efficiency in mind as well as a flexible data model for integrating with existing software, Viskores provides performant execution on modern compute architectures as demanded by in-situ applications.

The Viskores software library has brought together algorithms from many independent researchers and provided them in a form that makes it accessible to users and other software. This paper provides a short history of Viskores and related visualization work (Section 2), a brief overview of Viskores features (Section 3), an overview of Viskores' data representation and how it can adapt to data from other software (Section 4), how Viskores manages the interface between runtime dynamic CPU types and static GPU types (Section 5), and finally an overview of software already integrated with Viskores (Section 6).

2. Related Work

The Viskores library inherits software from a long line of research and development. As GPUs capable of general-purpose computing became more common around 2010, the scientific visualization research community began investigating techniques to leverage these processors. Early research was fragmented into separate libraries that specialized in the main focus of the group developing them: portable algorithms [LSA12], data structures [MSPA12], and program abstractions [MKMM12]. To address the growing need to operate scientific visualization on GPUs and other similar processors, a DOE ASCR research project [MPR*19] combined these techniques into a single software library named VTK-m [MSU*16]. VTK-m was designed as a general-purpose model to implement scientific visualization on accelerator architectures (predominantly GPUs today) that worked across programming models in a performance-portable way. VTK-m later became part of the Exascale Computing Project (ECP) to enable scientific visualization as the first exascale supercomputers became available [MAB*24].

Viskores is the successor of VTK-m. Although not a drop-in replacement, the first release of Viskores mimics the structure of VTK-m such that provided scripting tools can convert existing software to the new library through symbol renaming. Viskores was launched as one of the founding projects of the High Performance Software Foundation (HPSF) [HPS24]. HPSF is an umbrella group of the Linux Foundation dedicated to lowering barriers to high-performance computing, and it assists Viskores with maintenance, support and outreach.

Although it fills a niche in scientific visualization, Viskores joins other scientific visualization libraries and applications. Likely the most well known and widely used scientific visualization library is VTK [SML04]. This open-source product has had contributions spanning over thirty years [SG24] and remains an active development community. This makes VTK a comprehensive collection of scientific visualization algorithms and features although its code structure primarily limits its implementation to CPU processors. Because there has been such a large investment in the VTK software, the Viskores approach has been to augment rather than replace its capabilities, which both improves existing software and helps Viskores have a larger practical impact. Some secondary libraries such as PyVista [SK19] and vedo [Mus*] use VTK to provide simplified programming interfaces, and many applications such as ParaView [AGL05], VisIt [CBW*12], and Mayavi [RV11] use VTK to implement scientific visualization in an end-user interface. Like Viskores, other related libraries provide complementary features. For example, ITK [IS03] provides 3D imaging filters frequently used in related medical visualizations and TTK [TFL*18] provides topology algorithms. As the names imply, these toolkits are frequently used together with VTK. Other alternate libraries such as yt [TSO*11] and VisPy [Cam*] feature lightweight scientific visualization at the expense of fewer capabilities.

Viskores was never intended to supplant any of these libraries. Rather, Viskores provides unique, accelerated algorithms that are meant to be integrated with other software. This paper focuses on how the design of Viskores allows it to operate with other libraries and applications.

3. Viskores Features

Although designed to operate on data using GPUs and other accelerators, Viskores provides an outwardly facing interface of standard C++ function and method calls. The upshot is that software projects using Viskores libraries do not need to adopt the device-specific compilers, language extensions, and optimization flags required when building Viskores itself. Viskores filters provide the interface that closes the gap between an algorithm studied in a research paper and a software product using the algorithm.

Available through this simplified interface is a collection of *filters*, which operate on one or more dataset to produce a new dataset derived from this input. Also available is a self-contained *rendering* library that produces a visual representation of a dataset object in an image. These features are described in Sections 3.1 and 3.2, respectively. Viskores also provides a small I/O library for reading and writing files. However, this library is primarily provided for learning and testing purposes. For production uses of Viskores, data is adapted from other sources using the representations documented in Section 4 or through a secondary library that has already done this adaption such as those described in Section 6.

3.1. Filters

Viskores provides a broad collection of accelerated, data-parallel filters organized into modular components. Each module corresponds to a coherent family of algorithms and is built as an independent CMake target, allowing applications to enable and link only the functionality they require. This design improves maintain-

ability, reduces build and link complexity, limits binary size, and simplifies long-term evolution of the code base.

3.1.1. Fundamental Modules

The provided filters are grouped by algorithmic families. Many of these families are fundamental to scientific visualization operation that have been updated for accelerated processors.

Clean grid operations can improve structural problems by removing degenerate cells, fusing coincident points [YCM19], and converting the representation to a common, explicit structure. *Connected component* operations identify groups of cells that are connected by any existing path [SLH*15] defined by either connections of the cells or, for regular grids, neighbors with a common classification.

The *contour* module implements multiple parallel isosurface extraction algorithms, including *marching cells* for general unstructured meshes and *flying edges* for structured and uniform grids. Marching cells generalizes marching cubes [LC87; LSA12; MSU*16] and marching tetrahedra to arbitrary cell types and is applicable to explicit and other unstructured cell sets, whereas flying edges [SMG15] exploits the regular memory layout of uniform and image grids to minimize memory traffic and improve parallel scalability. The general `Contour` filter uses one of these algorithms, depending on the data structure, to generate level sets from a field in the data. The `Slice` filter provides a convenient interface to find the contours of implicit functions evaluated on spatial coordinates. `Clip` [TSSM26a] is an alternate form of contouring that extracts the region of cells meeting a specified field range or spatial region.

Viskores also provides a *density estimate* module for computing statistical summaries and distribution-based analyses of scalar and multivariate data. These filters include `Histogram` and `NDHistogram` for one- and multi-dimensional binning, `Entropy` and `NDEntropy` for information-theoretic measures, and particle-based estimators such as `ParticleDensityNearestGridPoint` and `ParticleDensityCloudInCell`. Complementing these are descriptive statistics filters that compute common statistical measures (e.g., minimum, maximum, mean, and variance) in parallel over large data sets, enabling scalable quantitative analysis in both in situ and post hoc workflows.

Beyond these, Viskores provides additional mesh- and field-oriented capabilities. The *entity extraction* module includes representative filters such as `Threshold` [MMA*13] and `ExternalFaces` [LBMC16; LMLC17; LLC20; TSSM26b] for subsetting and boundary extraction. The *field conversion* and *field transform* modules support operations such as `CellAverage`, `PointAverage`, `CoordinateSystemTransform`, and `WarpScalar`, enabling transfer and transformation of field data across mesh representations. *Mesh information* (e.g., `MeshQuality`) provides information about each cell in a dataset. Filters in the *Geometry refinement* module convert the structure of cells in various ways such as dividing cells into simplices, coarsening a mesh [MSU*16], or converting the mesh to a point cloud.

3.1.2. Flow Analysis

Central to the *flow* module is particle advection [PYK*18]. This finds the path a particle takes based on its vector field, and these particle traces form the basis of our flow algorithms. Streamlines and pathlines show the curve traced particles travel in steady and unsteady flow, respectively. Stream surfaces [Hul92] extend streamlines to find the surface flowing from an originating curve. Lagrangian coherent structures (LCS) find regions where the flow converges and diverges. These are computed in Viskores by computing a finite time Lyapunov exponent (FTLE), which requires advecting particles over the entire mesh [SYB*21]. This particle advection has also been leveraged to accelerate the tracing of charged plasma particles following the Lorentz-Force in laser wakefields [MAB*24] and the tracing of field lines in a tokamak magnetic confinement field to generate Poincaré puncture maps [PCK*24].

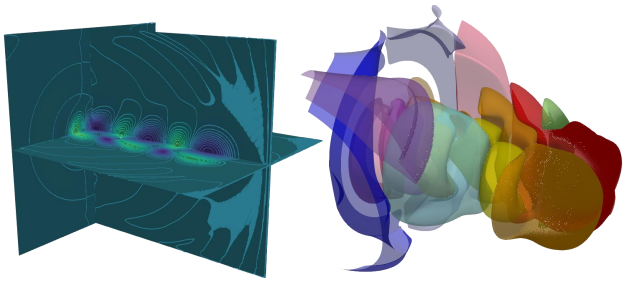

Figure 2: *Contour-tree–based analysis applied to a 3D WarpX laser-driven, plasma-based particle accelerator simulation dataset. Left: Slices with extracted contours on the slice. Right: The top 11 branches of the contour tree. (Image reproduced from Li et al. [LCR*25a] with permission by IEEE.)*

3.1.3. Topology Filters

Viskores provides a *scalar topology* module for topological analysis of scalar fields. A contour tree summarizes the evolution of isosurfaces as the scalar isovalue varies, capturing how connected components merge and split. As such, it provides a compact representation of global topological structure and supports feature identification, hierarchical simplification, segmentation, and quantitative analysis of scalar data.

The `ContourTreeUniformAugmented` filter implements the data-parallel peak pruning (PPP) algorithm [CWS*21]. To efficiently support augmentation and queries, the implementation uses the hyperstructure, a hierarchical representation that enables logarithmic-time searches and parallel insertion of regular vertices [CRWA22]. For distributed-memory environments, the `ContourTreeUniformDistributed` filter implements distributed hierarchical contour trees [CRW22]. The filter computes local contour trees with optional pre-simplification [LCR*25b] in parallel and exchanges only boundary information during a logarithmic fan-in process.

Several filters support data analysis based on the distributed contour tree [LCR*25a]. The `DistributedBranchDecompositionFilter` computes the branch decomposition of the

distributed contour tree, that is, a hierarchical representation that ranks features (isosurface components) by importance. Currently, the filter uses contour volume as the importance metric. Based on the branch decomposition, the `SelectTopVolumeBranchesFilter` identifies the most relevant connected components of an isosurface, that is, the connected components with the largest volumes, and the `ExtractTopVolumeContoursFilter` extracts the corresponding connected components of the isosurface and supports their visualization. Figure 2 shows an illustration of applying these filters to a data set resulting from a WarpX simulation.

3.1.4. Uncertainty Visualization

Several new techniques for visualization of uncertainty in 2D and 3D scientific data have been developed in the past decade to reduce misrepresentation and increase trust in results [BOL12; PRJ12; WHLS19; KDJ*21]. However, the computational expense and overhead associated with uncertainty propagation has limited their use in practical visualization tools. Fortunately, independent computations within uncertainty visualization algorithms can be identified, making them well-suited for parallel execution [WAM*23; AJSP23; AWP*25; AMP*26]. As a result, these algorithms can be accelerated using Viskores to achieve interactive performance, thereby helping bridge the gap between uncertainty visualization research and practical visualization tools.

Viskores is among the first parallel libraries to provide built-in support for uncertainty visualization. In particular, it offers the `ContourUncertainUniform` filter for visualizing positional uncertainty in 3D isosurfaces of univariate scalar data [ASJ21; PH11] derived from the marching cubes algorithm [LC87]. Additionally, Viskores includes a `FiberUncertainUniform` filter for uncertainty visualization of fibers [ZS21; AJSP23; HJW*24], an extension of the level-set concept for univariate to bivariate scalar data [CGT*15]. We envision these capabilities being adopted by the broader scientific community to facilitate uncertainty-aware analysis of their complex data.

3.2. Rendering

Modern HPC workflows run on leadership-class supercomputers like Frontier, which do not have a system-level rendering context. Throughout their execution, simulation data stays in accelerator memory. Viskores addresses both constraints with a lightweight, device-portable rendering pipeline that executes on the same device and requires no system graphics context. The pipeline organizes rendering through a `Scene` / `Actor` / `Mapper` / `Canvas` decomposition, inspired from VTK but redesigned for device-resident execution.

Each `Actor` binds a `DataSet` to a `Mapper` that determines how the data is drawn. Multiple rendering algorithms are provided through specialized mapper classes. `MapperVolume` operates only on structured grids and composites fragments back-to-front for volume transparency, whereas `MapperRayTracer` accepts any cell topology and traverses the scene front-to-back for early ray termination. It implements data-parallel ray casting [LLN*15; LMNC15]. Additional mappers handle point clouds, mesh wireframes, glyphs, quad surfaces, and cylindrical geometry. Beyond color-mapped rendering, `MapperConnectivity` also supports simulated radiography, integrating absorption and emission across multiple spectral energy bins (over 100 supported) to model radiation propagation through simulation volumes.

For distributed-memory workflows, systems such as Ascent extend this pipeline to leadership-class scale by coordinating per-rank Viskores rendering and handling cross-node image compositing.

For applications with an existing renderer, Viskores provides interoperability with ANARI [SGA*22], a Khronos standard that decouples applications from specific rendering backend implementations as described in Section 6.4. Viskores stages geometry and field data into ANARI-compatible representations, enabling compatibility with any ANARI backend.

4. Data Representation

Key to the integration of Viskores with other software products is its representation of data. Algorithms created for research often use specific data structures, but the Viskores data representations allow the same algorithms to apply across a plethora of structures. Viskores structures data in a way that is flexible enough to adapt to data structures defined in external software libraries but still afford efficient access while running filters and other features. This section summarizes how data is managed in Viskores and generally how Viskores shares data with other software.

4.1. Mesh Representation

Like VTK [SML04], Viskores primarily opeates on mesh structures comprising a 3D space divided into *cells*. Cells can be 0D (vertex), 1D (line), 2D (polygon), or 3D (polyhedral) regions that are defined by a collection of points that are connected in a predefined way prescribed by the *shape* of the cell. Cell adjacency is defined by cells sharing points with the same indices. A Viskores `DataSet` object defines a mesh with three structures.

Cell Set The `DataSet` holds a single cell set object that defines for each cell its shape and the points it is incident on. The cell set object is polymorphic, so the representation of the cell structure can adapt to the data. Viskores provides multiple cell set objects, but cells are usually defined as either a regular grid of cells or with arrays explicitly specifying incident points.

Fields The `DataSet` holds a collection of arrays that provide a mapping from a topological element in the cell set to a value. Each field is represented as an array that is associated with a type of topological element such as points or cells. The field provides a lookup from the index of the element to a value.

Coordinates The `DataSet` identifies one or more fields as point coordinates. Coordinate fields map points to 3D vectors. These define the physical location of the geometry, which has special significance in many Viskores operations.

Unlike VTK, Viskores does not define the coordinate type as part of the cell structure. This provides more flexibility in defining data sets [MSPA12].

Viskores also provides a `PartitionedDataSet` to hold collections of `DataSet` objects that together form assemblies or hierarchies.

The data set objects provide structure to data provided in simple arrays. When interfacing with other software products, memory is communicated through these arrays. The following sections describe how Viskores manages arrays and how they can work with external memory.

4.2. Array Representation

Viskores manages arrays using a class called `ArrayHandle`. This is a templated class with the first template parameter specifying the type of each value in the array and the second parameter specifying how the values are stored in memory using a secondary object in a strategy pattern [GHJV95]. This allows the single `ArrayHandle` class to manage every type of array in Viskores and likewise for every algorithm in Viskores to specialize to any representation of data. Array storage does not have to be limited to explicit values in memory. This mechanism also allows for arrays to be transformed into others or for implicit arrays to use no memory and evaluate each value on the fly.

For an `ArrayHandle` to be used directly, its full type with template parameters must be known. To allow objects like `DataSet` to store references to arrays without specifying their full type, Viskores provides `UnknownArrayHandle`, which holds a reference to an `ArrayHandle` without knowing its type. Arrays in `UnknownArrayHandle` cannot be accessed directly. They must first be converted back to an `ArrayHandle`, a challenge discussed in Section 5.

Depending on how its data is stored, an `ArrayHandle` may need to manage data in multiple separate sections of memory. To simplify this management, Viskores uses a separate `Buffer` object.

4.3. Buffer Representation

A `Buffer` object manages a single, contiguous region of memory. It manages how the memory is accessed on the host (CPU) and device (GPU), which can be in different memory spaces. The `Buffer` object manages the allocation and sizing of the buffer as well as in what memory spaces it is available. Objects like `ArrayHandle` can request a reference to the memory of the buffer on a particular device, and the `Buffer` will ensure the data is available in that memory space.

The `Buffer` object is the keystone for sharing data with Viskores. External software can create a `Buffer` using memory that it provides. The `Buffer` can be given a device on which the memory resides so, for example, data born on a GPU can be passed to Viskores without leaving. The `Buffer` can also be provided with functions to destroy or reallocate the memory so Viskores memory management can be reflected back in the originating software.

Thus, arrays are zero-copied into Viskores by constructing `Buffer` objects using these arrays and then constructing `ArrayHandles` of the matching layout with these buffers. Likewise, data can be pulled zero-copy by accessing the `Buffers` in an `ArrayHandle`. Viskores has helper methods to move data in and out of `ArrayHandles` of common formats.

5. Managing Structure Polymorphism

As described in Section 4, a `DataSet` represents mesh structures with a meaningful collection of arrays, and these arrays are stored polymorphically such that the type of the array can remain unspecified and queried during program execution. However, repeatedly accessing array values through a polymorphic function is generally discouraged, particularly on a GPU [ZAR21]. Thus, Viskores requires that an unknown array must be cast back to an `ArrayHandle` with its template parameters specified before the values can be accessed.

If an `UnknownArrayHandle` has a specific expected type, then it can be cast to the appropriate known `ArrayHandle`. Most frequently, the type of an `UnknownArrayHandle` must be queried and cast accordingly.

5.1. Strategies for Casting Unknown Arrays

Viskores uses three general approaches for pulling a statically typed array from an unknown array container: static type visitation, strided component reference, and an array copy fallback. In our experience, using all three together is best to keep the number of type checks tractable.

5.1.1. Static Type Visitation

Viskores provides several forms of *cast-and-call* functions. They take an abstract object, a list of types, and a templated function; they test the object against each type, and call the provided function with the proper downcast of the object. This provides a static type visitation [Dav04] similar to the C++ STL `std::visit` function.

When using static type visitation, the compiler must create an instance of the target function for every type checked regardless of whether that form of the function is ever called. The type visitation is usually done in the host code, which causes a separate device kernel to be created for each type instance. Is is also possible to push the static type visitation into the device code through multiplexer objects. For example, point coordinates can always be treated as 3-component float vectors, so an array multiplexer can be used to pull such values out of a collection of array types. Although this approach can reduce the number of kernel instances generated, the switch statements required for the multiplexer operation can slow down GPU kernels.

Ideally, type visitation would dispatch code for every type an algorithm would operate on. Unfortunately, this is not viable because it would result in too many kernel instances. A single array could contain values of several base types (`float`, `double`, `int`, etc.) and each of which could be constructed into vectors of different sizes. Additionally, such data can be lain out as a sequence of vectors (array of structures) or with each component in a separate buffer (structure of arrays) or numerous other specialty representations. Consequently, there may be dozens of types to represent a single array. When algorithms require two or more arrays of input, the cross product of type combinations can explode into hundreds or even thousands of combinations, which causes problems for compilation. Thus, in practice the number of types checked must be reduced, which leads to the remaining strategies.

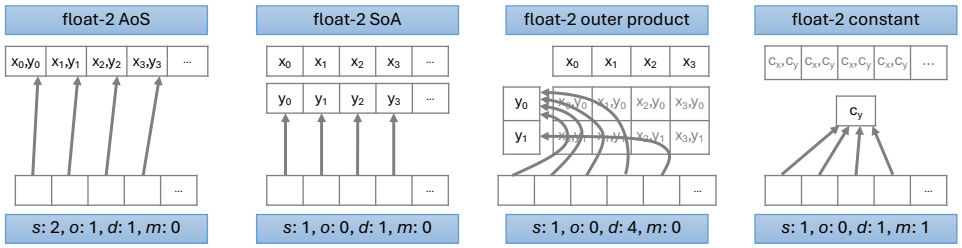

Figure 3: *Demonstration of representing different memory layouts with the* `ArrayHandleStride` *parameters.*

5.1.2. Strided Component Reference

Viskores provides an array named `ArrayHandleStride` that allows memory stored in a single buffer to be referenced in a flexible way. The stride array handle treats its buffer as a basic list of values but potentially skips over entries in the array. Its state contains an offset to skip over the first o values in the buffer and a stride to increment s times for each index of the array. It can also repeat values in the buffer with an integer divisor that effectively repeats each entry d times and a modulus that repeats all values after m entries. `ArrayHandleStride` converts an index to its logical array i_{in} to an index of the buffer array i_{buf} with Equation 1.

$$i_{buf} = (((i_{in} \bmod m) \operatorname{div} d) \cdot s) + o \tag{1}$$

`ArrayHandleStride` has a special purpose in Viskores; it can be used to reference one of the components of vector arrays in several common memory layouts as demonstrated in Figure 3. If vectors are arranged in a basic, packed layout in a single buffer (an array of structures), components are referenced by setting the offset, o, to the component index and the stride, s, to the number of components. If vectors are arranged with each component in a separate buffer (a structure of arrays), components are easily referenced from that component buffer with $o = 0$ and $s = 1$. The divisor and modulus allow for more complicated arrangements like the Cartesian product describing the point coordinates of irregularly spaced axis-aligned grids. Even arrays of a constant value can be represented by putting a single value in a buffer and using $m = 1$ to always repeat the first value.

`UnknownArrayHandle` provides a mechanism to "extract" a component represented in an `ArrayHandleStride` knowing only the single component type. Although the array indexing arithmetic does add some overhead, referencing array components this way allows algorithms to operate on arrays without compiling for specific memory storage patterns or the number of components in a vector. This dramatically reduces the number of static types to visit from the technique in Section 5.1.1.

Because algorithms commonly must operate on all components of an array at once, Viskores can also extract all components at once and collect the `ArrayHandleStride` objects into a pseudo-array that provides access to all components at once regardless of how many components there are. It should be noted that referencing arrays this way does add complications to algorithm design. Because the size of each vector in the array is not known at compile time and Viskores does not currently support allocating scratch memory per thread, algorithm implementations cannot generate intermediate vector values. Operations must be done by iterating over components or using the space reserved for output values as scratch space before writing the final value.

5.1.3. Array Copy Fallback

Often, a Viskores algorithm can make reasonable assumptions about the the type of array it is processing without being certain. For example, most Viskores filters assume that a point field is represented by floating point values as interpolation through cells only makes sense with real numbers. However, it is conceivable that values were generated with fixed-point precision and represented as integer numbers. Likewise, a filter may make assumptions of one field type from another. For example, a filter operating on uncertain data (such as that described in Section 3.1.4) may determine the type of an array of mean values and assume a companion array of deviations has the same type. Although it is reasonable to assume that these two arrays were generated together and with the same type, there is no guarantee this is the case.

Assumptions of this nature simplify implementation and keep template specialization to a manageable level, but filters should not fail when these assumptions are broken. To prevent such failures, Viskores provides a simple fallback: copying the array to a known type. Viskores' `ArrayCopy` method copies the data from an unknown type to a known type. This allows a filter to, for example, check whether a field has 32-bit or 64-bit floats and, if it has neither, copy to an array of floats and operate as if it had. The `ArrayCopyShallowIfPossible` function simplifies implementing this fallback by passing a reference to the array if it matches the specified type or copying the data if it does not. In the uncertainty filter example above, the filter can construct an array of the same type as the mean array and then use `ArrayCopyShallowIfPossible` to get the deviation values.

Copying arrays has obvious time and memory overhead. However, as long as all expected array types have specializations, the copy fallback well be seldom if ever used. Although copying an array is not ideal, it is better than an outright failure.

5.2. Filter Base Class Field Handling

Because most algorithms in Viskores are implemented as filters, the `viskores::filter::Filter` base class contains features to simplify implementing the strategies discussed in Section 5.1. These are in the form of custom cast-and-call methods

that wrap static type visitation, strided component reference, and array copy fallback together.

The first such method is `CastAndCallScalarField`. This behavies like a basic static type visitation, but internally also attempts strided component reference where needed and has an array copy fallback. In addition to simplifying the code, this allows Viskores to internally define what common types are expected in scalar fields, allow for customization in compile configuration options, and broadly apply these across filter algorithms.

A similar method, `CastAndCallVecField` provides a comparable feature for vector fields. For a specified vector size, this method will similarly perform static type visitation for expected types with component references and a copy fallback to avoid excessive callbacks. For cases where the size of the vector is not fixed, the `CastAndCallVariableVecField` uses strided component reference to pull a number of components defined at runtime.

At the completion of its algorithm, a filter must build a `DataSet` encapsulating the results. Part of this building process includes the conversion of fields from the input data, many of which had no effect on the filter's algorithm, to the output data. The `Filter` base class provides several forms of a `CreateResult` method to assist in the creation of this `DataSet` and the conversion of these fields.

In cases where the geometry of the filter is not modified, the field arrays can be simply moved from input to output, and `CreateResult` will do this automatically in this case. In cases where the geometry is changed, `CreateResult` can invoke a callback to the filter implementation to map each field from input to output. Again, Viskores provides helper functions for common patterns. For cases where each output value comes from a particular input value, `MapFieldPermutation` will perform this value reordering from a permutation array without the filter determining the field array type. This covers cases like the threshold operation where values are removed and cases like triangulation where values are added. `MapFieldMergeAverage` provides another common pattern where values are grouped by keys, and grouped values are averaged together. This covers cases like coincident point merging. The permutation array or keys required for each of these two functions can usually be reused from scattering operations [MMP*21] generated as part of the filter's algorithm.

6. Integration

As one of the founding software projects of the HPSF, Viskores has a commitment to supporting an operational, open-source HPC software stack. Activities supporting this goal include automated continuous integration and other good software development practices, regular software maintenance and release, and deployment through package managers like Spack [GLC*15]. As a support library, part of Viskores' role in the HPC software stack is its integration with applications and other libraries. In this section, we review some of the software packages featureing integration with Viskores.

6.1. VTK

VTK and Viskores are closely related but serve distinct roles within the visualization ecosystem. Whereas VTK provides a broad, end-to-end framework encompassing data models, filters, rendering, and I/O, Viskores focuses specifically on performance-portable, data-parallel algorithms designed to run efficiently on various many-core architectures. Rather than replacing VTK, Viskores complements it by providing high-performance primitives and filters that can be integrated directly into the VTK pipeline.

6.1.1. Array Exposure and Execution Model

A key aspect of this integration is zero-copy data interoperability. The VTK arrays underlying VTK data objects can be seamlessly wrapped as native Viskores arrays. Conversely, Viskores arrays can be wrapped as VTK arrays via the `vtkmDataArray` class, enabling VTK algorithms to operate directly on Viskores data without redundant memory allocation or data movement. Notably, when converting back to VTK, native VTK array types are intentionally not used to preserve correct memory management, particularly for GPU-resident data. This interoperability supports both CPU- and GPU-resident memory, as well as a wide range of array types, including both explicit and implicit arrays.

6.1.2. Filters and Pipeline Integration

Viskores filters are exposed within VTK as adapter filters. These adapters manage the communication between the two systems: they convert VTK datasets into Viskores equivalents (using zero-copy sharing where possible), execute the corresponding Viskores algorithm when constraints are met, and then convert the results back into VTK datasets. In many cases, these Viskores-backed filters can override native VTK implementations. This allows VTK to preserve its traditional pipeline semantics while transparently gaining the performance benefits of Viskores.

6.2. Conduit and Ascent

Ascent is a lightweight in situ visualization and analysis library designed to be embedded directly into large scale simulation codes [LBCH22]. Simulations interact with Ascent by publishing their data using Conduit's mesh blueprint at selected timesteps or intervals. The application uses the Conduit library to describe the simulation's meshes, fields, and associated metadata [HLR*22].

Viskores is a core component of Ascent's visualization and analysis capabilities to guarantee shared-memory portability and performance on a variety of backends whereas Ascent itself is responsible for the distributed-memory orchestration using MPI. Ascent converts each domain's Conduit blueprint description into a Viskores `DataSet`. Then, each MPI rank operates on its local collection of `DataSet`s, runs the requested Viskores filters or rendering, and participates in subsequent MPI-based coordination (e.g., reductions or image compositing) required by the overall in situ workflow.

Conduit's zero copy data transport allows simulations to expose CPU or GPU resident fields in mesh blueprint form without moving or duplicating the data. Ascent can then map this blueprint representation directly into Viskores datasets so Viskores filters and renderer execute on the original simulation data.

6.3. ADIOS

Large-scale scientific workflows require scalable mechanisms for moving data between simulations, storage, and downstream analysis. ADIOS [GPW*20] provides this data transport layer. Designed for leadership-class systems, ADIOS abstracts data movement through modular engines that support high-performance file-based I/O plus in situ and in transit coupling [EPG*25].

ADIOS has enabled in situ and in transit visualization and analysis using Viskores across a variety of use cases. These include scaling studies to analyze performance of in situ techniques [KLC*20a; PCK*24; KLC*20b; MGG23] and production simulation codes [CCD*18; BBC*13; E3S18; KT02; WHW16]. In these deployments, simulations publish mesh and field data through ADIOS streams, and Viskores-based analysis components consume the data directly—either concurrently with the simulation or as loosely coupled in transit services. This integration allows applications to incorporate accelerator-portable visualization and analysis without disruptive changes to software structure.

6.3.1. Fides: A schema for Viskores DataSets

ADIOS provides efficient data transport but does not define mesh or field semantics. Scientific visualization requires the structured representations described in Section 4. Viskores addresses this gap through *Fides* [PRT*21], a schema-driven interface that interprets ADIOS variables as structured scientific datasets.

Fides maps ADIOS-managed arrays into Viskores `DataSets` according to a declarative description in JSON. This approach separates physical data layout from semantic interpretation, allowing simulations to write raw arrays while external schema descriptions define how those arrays form meshes. This design preserves Viskores' performance-portable execution model while enabling flexible, schema-driven interoperability.

6.4. ANARI

ANARI is a portable 3D rendering API maintained by the Khronos Group [SGA*22]. ANARI connects applications from diverse domains to any 3D rendering engine implementing ANARI while still giving vendors many degrees of freedom of how exactly rendering is done.

Viskores leverages ANARI in two key ways: providing mappers for translating datasets to ANARI objects and implementing a Viskores-based renderer.

6.4.1. ANARI Mappers

Viskores provides an interop library containing mapper classes that translate Viskores data into renderable ANARI objects. Applications wrap dataset objects and provide them to a *mapper*, which turns the selected data into a renderable representation, such as a volume, point cloud, or triangle mesh. The resulting ANARI object from the mapper is then placed into an `ANARIWorld` by the application, who finally uses ANARI directly for configuring the rest of the scene (additional scene objects, camera, renderer, and frame) to ultimately render images.

6.4.2. ANARI Device Implementation

ANARI calls instances of 3D engines *devices*, and multiple devices can be used concurrently in applications. Viskores provides an ANARI device implementation using its internal raycasting capabilities. Although other high-quality, accelerated implementations (such as OSPRay [WJA*17], VisRTX [Ams25], and Barney [WZA*24]) are expected to be used when possible, this implementation exists to give Viskores users a simple, consistent renderer that is always available on platforms supported by Viskores itself. Additionally, other existing ANARI applications are now able to use Viskores' rendering capabilities without needing to do an explicit Viskores integration.

The core renderer design implements a simple raycast renderer through a multi-pass approach. `ANARISurface` and `ANARIVolume` objects are internally represented as Viskores datasets and iteratively rendered. All surfaces are rendered first, followed by volumes, creating a coherent single image of the entire scene.

7. Conclusion

Viskores has key capabilities that has enabled adoption in varied settings. As described throughout the paper, it supports numerous algorithms, many data representations, and does so with efficient memory usage and execution, including on GPU architectures. This approach has filled a clear need within the community, seeing adoption in VTK, Ascent, ADIOS, and ANARI. Going forward, the developers of Viskores endeavor to continue adding capabilities to meet diverse requirements, and to ensure that Viskores is maintained and performant as hardware continues to be evolved.

In particular, AI and ML are of growing interest to the scientific and visualization communities [HBL*19; LKP19; LCBP23; MYLP24; HHP*24; ATW26; ZSMG26]. It is unclear what role AI will play in Viskores or vice versa, but as a first step we look forward to integrating Viskores arrays with the tensors used in AI libraries. This data sharing will make possible the blending of new AI techniques with traditional scientific visualization approaches.

8. Acknowledgements

The authors wish to express their gratitude to all contributors to Viskores, VTK-m, and other predecessors products that made Viskores possible.

This material is based upon work supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, Next-Generation Scientific Software Technologies program and Early Career Research Program, under contract number DE-AC05-00OR22725. This research used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725. This research also used resources of the ALCF, a DOE Office of Science user facility at Argonne National Laboratory and is based on research supported by the DOE Office of Science-Advanced Scientific Computing Research Program under contract number DE-AC02-06CH11357.

References

[AGL05] AHRENS, JAMES, GEVECI, BERK, and LAW, CHARLES. "ParaView: An End-User Tool for Large Data Visualization". *Visualization Handbook*. ISBN 978-0123875822. Elesvier, 2005. DOI: 10.1016/B978-012387582-2/50038-1 2.

[AJSP23] ATHAWALE, TUSHAR M., JOHNSON, CHRIS R., SANE, SUDHANSHU, and PUGMIRE, DAVID. "Fiber Uncertainty Visualization for Bivariate Data With Parametric and Nonparametric Noise Models". *IEEE Transactions on Visualization and Computer Graphics* 29.1 (2023), 613–623. DOI: 10.1109/TVCG.2022.3209424 4.

[AMP*26] ATHAWALE, TUSHAR M., MORELAND, KENNETH, PUGMIRE, DAVID, et al. "MAGIC: Marching Cubes Isosurface Uncertainty Visualization for Gaussian Uncertain Data with Spatial Correlation". *IEEE Transactions on Visualization and Computer Graphics* (2026), 1–16. DOI: 10.1109/TVCG.2026.3653244 4.

[Ams25] AMSTUTZ, JEFFERSON. *VisRTX: A NVidia OptiX based implementationof ANARI*. 2025. URL: https://github.com/NVIDIA/VisRTX 8.

[ASJ21] ATHAWALE, TUSHAR M., SANE, SUDHANSHU, and JOHNSON, CHRIS R. "Uncertainty Visualization of the Marching Squares and Marching Cubes Topology Cases". *IEEE VIS*. 2021, 106–110. DOI: 10.1109/VIS49827.2021.9623267 4.

[ATW26] AI, KUANGSHI, TANG, KAIYUAN, and WANG, CHAOLI. "NLI4VolVis: Natural Language Interaction for Volume Visualization via LLM Multi-Agents and Editable 3D Gaussian Splatting". *IEEE Transactions on Visualization & Computer Graphics* 32.1 (Jan. 2026), 46–56. DOI: 10.1109/TVCG.2025.3633888 8.

[AWP*25] ATHAWALE, TUSHAR M., WANG, ZHE, PUGMIRE, DAVID, et al. "Uncertainty Visualization of Critical Points of 2D Scalar Fields for Parametric and Nonparametric Probabilistic Models". *IEEE Transactions on Visualization and Computer Graphics* 31.1 (2025), 108–118. DOI: 10.1109/TVCG.2024.3456393 4.

[BBC*13] BUSSMANN, M., BURAU, H., COWAN, T. E., et al. "Radiative signatures of the relativistic Kelvin-Helmholtz instability". *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. SC '13. Association for Computing Machinery, 2013. DOI: 10.1145/2503210.2504564 8.

[BOL12] BRODLIE, KEN, OSORIO, RODOLFO ALLENDES, and LOPES, ADRIANO. "A Review of Uncertainty in Data Visualization". *Expanding the Frontiers of Visual Analytics and Visualization*. Ed. by DILL, JOHN, EARNSHAW, RAE, KASIK, DAVID, et al. Springer Verlag London, 2012, 81–109. DOI: 10.1007/978-1-4471-2804-5_6 4.

[Cam*] CAMPAGNOLA, LUKE et al. *VisPy*. Zenodo. DOI: 10.5281/zenodo.592490 2.

[CBW*12] CHILDS, HANK, BRUGGER, ERIC, WHITLOCK, BRAD, et al. "VisIt: An End-User Tool For Visualizing and Analyzing Very Large Data". *High Performance Visualization: Enabling Extreme-Scale Scientific Insight*. CRC Press/Francis–Taylor Group, Oct. 2012, 357–372 2.

[CCD*18] CHOI, JONG YOUL, CHANG, CHOONG-SEOCK, DOMINSKI, JULIEN, et al. "Coupling exascale multiphysics applications: Methods and lessons learned". *2018 IEEE 14th International Conference on e-Science (e-Science)*. IEEE. 2018, 442–452. DOI: 10.1109/eScience.2018.00133 8.

[CGT*15] CARR, HAMISH, GENG, ZHAO, TIERNY, JULIEN, et al. "Fiber Surfaces: Generalizing Isosurfaces to Bivariate Data". *Computer Graphics Forum* 34.3 (July 2015), 241–250. DOI: https://doi.org/10.1111/cgf.12636 4.

[CRW22] CARR, HAMISH, RÜBEL, OLIVER, and WEBER, GUNTHER H. "Distributed Hierarchical Contour Trees". *Proceedings of the 12th IEEE Symposium on Large Data Analysis and Visualization (LDAV)*. Best paper award. Oct. 2022. DOI: 10.1109/LDAV57265.2022.9966394 3.

[CRWA22] CARR, HAMISH, RÜBEL, OLIVER, WEBER, GUNTHER H., and AHRENS, JAMES. "Optimization and Augmentation for Data Parallel Contour Trees". *IEEE Transactions on Visualization and Computer Graphics* 28.10 (2022). Published online 2021, 3471–3485. DOI: 10.1109/TVCG.2021.3064385 3.

[CWS*21] CARR, HAMISH A., WEBER, GUNTHER H., SEWELL, CHRISTOPHER M., et al. "Scalable Contour Tree Computation by Data Parallel Peak Pruning". *Transactions on Visualization and Computer Graphics* 27.4 (Apr. 2021). Published online 2019, 2437–2454. DOI: 10.1109/TVCG.2019.2948616 3.

[Dav04] DAVID ABRAHAMS, ALEKSEY GURTOVOY. *C++ Template Metaprogramming: Concepts, Tools, and Techniques from Boost and Beyond*. ISBN 0-321-22725-5. Addison Wesley Professional, Dec. 2004 5.

[E3S18] E3SM PROJECT, DOE. *Energy Exascale Earth System Model v1.0*. [Computer Software]. Apr. 2018. DOI: 10.11578/E3SM/dc.20180418.36 8.

[EPG*25] EISENHAUER, GREG, PODHORSZKI, NORBERT, GAINARU, ANA, et al. "Streaming Data in HPC Workflows Using ADIOS". *Proceedings of the Cray User Group*. Aug. 2025, 31–43. DOI: 10.1145/3725789.3725793 8.

[GHJV95] GAMMA, ERICH, HELM, RICHARD, JOHNSON, RALPH, and VLISSIDES, JOHN. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison Wesley, 1995 5.

[GLC*15] GAMBLIN, TODD, LEGENDRE, MATTHEW, COLLETTE, MICHAEL R., et al. "The Spack package manager: bringing order to HPC software chaos". *SC15: International Conference for High-Performance Computing, Networking, Storage and Analysis*. Nov. 2015, 1–12. DOI: 10.1145/2807591.2807623 7.

[GPW*20] GODOY, WILLIAM F., PODHORSZKI, NORBERT, WANG, RUONAN, et al. "ADIOS 2: The Adaptable Input Output System. A framework for high-performance data management". *SoftwareX* 12 (2020), 100561. ISSN: 2352-7110. DOI: https://doi.org/10.1016/j.softx.2020.100561 8.

[HBL*19] HU, KEVIN, BAKKER, MICHIEL A., LI, STEPHEN, et al. "VizML: A Machine Learning Approach to Visualization Recommendation". *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*. 2019, 1–12. DOI: 10.1145/3290605.3300358 8.

[HHP*24] HAMMER, JAMES, HOBSON, TANNER, PUGMIRE, DAVID, et al. "A Personalized AI Assistant For Intuition-Driven Visual Explorations". *IEEE 20th International Conference on e-Science (e-Science)*. Sept. 2024. DOI: 10.1109/e-Science62913.2024.10678681 8.

[HJW*24] HARI, GAUTAM, JOSHI, NRUSHAD, WANG, ZHE, et al. "FunM^{2}C: A Filter for Uncertainty Visualization of Multivariate Data on Multi-Core Devices". *2024 IEEE Workshop on Uncertainty Visualization: Applications, Techniques, Software, and Decision Frameworks*. 2024, 43–47. DOI: 10.1109/UncertaintyVisualization63963.2024.00010 4.

[HLR*22] HARRISON, C., LARSEN, M., RYUJIN, B. S., et al. "Conduit: A Successful Strategy for Describing and Sharing Data In Situ". *2022 IEEE/ACM International Workshop on In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization (ISAV)*. Los Alamitos, CA, USA: IEEE Computer Society, Nov. 2022. DOI: 10.1109/ISAV56555.2022.00006 7.

[HPS24] HPSF. *HPSF has Launched!* Blog. May 2024. URL: https://hpsf.io/blog/2024/hpsf-has-launched/ 2.

[Hul92] HULTQUIST, J. P. M. "Constructing stream surfaces in steady 3D vector fields". *Proceedings of the 3rd Conference on Visualization '92*. Oct. 1992, 171–178 3.

[IS03] IBÁÑEZ, LUIS and SCHROEDER, WILLIAM. *The ITK Software Guide*. ITK 2.4. ISBN 1-930934-15-7. Kitware Inc., 2003 2.

[KDJ*21] KAMAL, AASIM, DHAKAL, PARASHAR, JAVAID, AHMAD Y., et al. "Recent advances and challenges in uncertainty visualization: a survey". *Journal of Visualization* 24.5 (2021), 861–890. DOI: 10.1007/s12650-021-00755-1 4.

[KLC*20a] KRESS, JAMES, LARSEN, MATTHEW, CHOI, JONG, et al. "Comparing Time-to-Solution for In Situ Visualization Paradigms at Scale". *2020 IEEE 10th Symposium on Large Data Analysis and Visualization (LDAV)*. 2020, 22–26. DOI: 10.1109/LDAV51489.2020.00009 8.

[KLC*20b] KRESS, JAMES, LARSEN, MATTHEW, CHOI, JONG, et al. "Opportunities for Cost Savings with In-Transit Visualization". *High Performance Computing*. Springer International Publishing, 2020, 146–165. DOI: 10.1007/978-3-030-50743-5_8 8.

[KT02] KOMATITSCH, DIMITRI and TROMP, JEROEN. "Spectral-element simulations of global seismic wave propagation—I. Validation". *Geophysical Journal International* 149.2 (May 2002), 390–412. DOI: 10.1046/j.1365-246X.2002.01653.x 8.

[LBCH22] LARSEN, MATTHEW, BRUGGER, ERIC, CHILDS, HANK, and HARRISON, CYRUS. "Ascent: A Flyweight In Situ Library for Exascale Simulations". *In Situ Visualization For Computational Science*. Cham, Switzerland: Mathematics and Visualization book series from Springer Publishing, May 2022, 255–279 7.

[LBMC16] LESSLEY, BRENTON, BINYAHIB, ROBA, MAYNARD, ROBERT, and CHILDS, HANK. "External Facelist Calculation with Data-Parallel Primitives". *Eurographics Symposium on Parallel Graphics and Visualization (EGPGV)*. 2016. DOI: 10.2312/pgv.20161178 3.

[LC87] LORENSEN, WILLIAM E. and CLINE, HARVEY E. "Marching Cubes: A High Resolution 3D Surface Construction Algorithm". *SIGGRAPH Computer Graphics* 21.4 (Aug. 1987), 163–169. DOI: 10.1145/37402.37422 3, 4.

[LCBP23] LIU, LEI, CHEN, WEI-PENG, BAHRAMI, MEHDI, and PRASAD, MUKUL. "Automatic Generation of Visualizations for Machine Learning Pipelines". *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*. 2023, 1–5. DOI: 10.1145/3551349.3559504 8.

[LCR*25a] LI, MINGZHE, CARR, HAMISH, RÜBEL, OLIVER, et al. "Distributed Augmentation, Hypersweeps, and Branch Decomposition of Contour Trees for Scientific Exploration". *IEEE Transactions on Visualization and Computer Graphics (Proceedings IEEE VIS 2024)* 31.1 (Jan. 2025), 152–162. DOI: 10.1109/TVCG.2024.3456322 3.

[LCR*25b] LI, MINGZHE, CARR, HAMISH, RÜBEL, OLIVER, et al. "Extremely Scalable Distributed Computation of Contour Trees via Pre-Simplification". *Proceedings of the 15th IEEE Symposium on Large Data Analysis and Visualization (LDAV)*. Nov. 2025. DOI: 10.1109/LDAV68558.2025.00005 3.

[LKP19] LEVENTHAL, SAMUEL, KIM, MARK, and PUGMIRE, DAVID. "PAVE: An In Situ Framework for Scientific Visualization and Machine Learning Coupling". *2019 IEEE/ACM 5th International Workshop on Data Analysis and Reduction for Big Scientific Data (DRBSD-5)*. Nov. 2019, 8–15. DOI: 10.1109/DRBSD-549595.2019.00007 8.

[LLC20] LESSLEY, BRENTON, LI, SHAOMENG, and CHILDS, HANK. "HashFight: A Platform-Portable Hash Table for Multi-Core and Many-Core Architectures". *Electronic Imaging, Visualization and Data Analysis*. 2020, 376-1-376–13(13). DOI: 10.2352/ISSN.2470-1173.2020.1.VDA-376 3.

[LLN*15] LARSEN, MATTHEW, LABASAN, STEPHANIE, NAVRÁTIL, PAUL, et al. "Volume Rendering Via Data-Parallel Primitives". *Eurographics Symposium on Parallel Graphics and Visualization*. (Note: work initially performed in predecessor framework to VTK-m, and was subsequently ported to VTK-m.) 2015. DOI: 10.2312/pgv.20151155 4.

[LMLC17] LESSLEY, BRENTON, MORELAND, KENNETH, LARSEN, MATTHEW, and CHILDS, HANK. "Techniques for Data-Parallel Searching for Duplicate Elements". *IEEE Symposium on Large Data Analysis and Visualization (LDAV)*. 2017. DOI: 10.1109/LDAV.2017.8231845 3.

[LMNC15] LARSEN, MATTHEW, MEREDITH, JEREMY S., NAVRATIL, PAUL A., and CHILDS, HANK. "Ray Tracing Within a Data Parallel Framework". *IEEE Pacific Visualization Symposium (PacificVis)*. 2015, 279–286. DOI: 10.1109/PACIFICVIS.2015.7156388 4.

[LSA12] LO, LI-TA, SEWELL, CHRIS, and AHRENS, JAMES. "PISTON: A Portable Cross-Platform Framework for Data-Parallel Visualization Operators". *Eurographics Symposium on Parallel Graphics and Visualization*. 2012. DOI: 10.2312/EGPGV/EGPGV12/011-020 2, 3.

[MAB*24] MORELAND, KENNETH, ATHAWALE, TUSHAR M., BOLEA, VICENTE, et al. "Visualization at exascale: Making it all work with VTK-m". *The International Journal of High Performance Computing Applications* 38.5 (2024), 508–526. DOI: 10.1177/10943420241270969 2, 3.

[MGG23] MAZEN, FRANÇOIS, GIVORD, LUCAS, and GUEUNET, CHARLES. "Catalyst-ADIOS2: In Transit Analysis for Numerical Simulations Using Catalyst 2 API". *High Performance Computing*. Springer Nature Switzerland, 2023, 269–276. DOI: 10.1007/978-3-031-40843-4_20 8.

[MKMM12] MORELAND, KENNETH, KING, BRAD, MAYNARD, ROBERT, and MA, KWAN-LIU. "Flexible Analysis Software for Emerging Architectures". *2012 SC Companion (Petascale Data Analytics: Challenges and Opportunities)*. Nov. 2012, 821–826. DOI: 10.1109/SC.Companion.2012.115 2.

[MMA*13] MAYNARD, ROBERT, MORELAND, KENNETH, AYACHIT, UTKARSH, et al. "Optimizing Threshold for Extreme Scale Analysis". *Visualization and Data Analysis 2013, Proceedings of SPIE-IS&T Electronic Imaging*. 2013. DOI: 10.1117/12.2007320 3.

[MMP*21] MORELAND, KENNETH, MAYNARD, ROBERT, PUGMIRE, DAVID, et al. "Minimizing Development Costs for Efficient Many-Core Visualization Using MCD^3". *Parallel Computing* 108.102834 (Dec. 2021). DOI: 10.1016/j.parco.2021.102834 7.

[MPR*19] MORELAND, KENNETH, PUGMIRE, DAVID, ROGERS, DAVID, et al. *XVis: Visualization for the Extreme-Scale Scientific Computation Ecosystem, Final Report*. Tech. rep. SAND 2019-9297. Sandia National Laboratories, Aug. 2019. DOI: 10.2172/1762947 2.

[MSPA12] MEREDITH, JEREMY S., SISNEROS, ROBERT, PUGMIRE, DAVID, and AHERN, SEAN. "A Distributed Data-Parallel Framework for Analysis and Visualization Algorithm Development". *Proceedings of the 5th Annual Workshop on General Purpose Processing with Graphics Processing Units (GPGPU-5)*. Mar. 2012, 11–19. DOI: 10.1145/2159430.2159432 2, 4.

[MSU*16] MORELAND, KENNETH, SEWELL, CHRISTOPHER, USHER, WILLIAM, et al. "VTK-m: Accelerating the Visualization Toolkit for Massively Threaded Architectures". *IEEE Computer Graphics and Applications* 36.3 (May 2016), 48–58. DOI: 10.1109/MCG.2016.48 2, 3.

[Mus*] MUSY, MARCO et al. *vedo, a python module for scientific analysis and visualization of 3D objects and point clouds*. Zenodo. DOI: 10.5281/zenodo.2561401 2.

[MYLP24] MALLICK, TANWI, YILDIZ, ORCUN, LENZ, DAVID, and PETERKA, TOM. "ChatVis: Automating Scientific Visualization with a Large Language Model". *SC24-W: Workshops of the International Conference for High Performance Computing, Networking, Storage and Analysis*. Nov. 2024, 49–55. DOI: 10.1109/SCW63240.2024.00014 8.

[PCK*24] PUGMIRE, DAVID, CHOI, JONG Y., KLASKY, SCOTT, et al. "Performance Improvements of Poincaré Analysis for Exascale Fusion Simulations". *VisGap - The Gap between Visualization Research and Visualization Software*. 2024. DOI: 10.2312/visgap.20241120 3, 8.

[PH11] PÖTHKOW, KAI and HEGE, HANS-CHRISTIAN. "Positional Uncertainty of Isocontours: Condition Analysis and Probabilistic Measures". *IEEE Transactions on Visualization and Computer Graphics* 17.10 (Oct. 2011), 1393–1406. ISSN: 1077-2626. DOI: 10.1109/TVCG.2010.247 4.

[PRJ12] POTTER, KRISTIN, ROSEN, PAUL, and JOHNSON, CHRIS R. "From Quantification to Visualization: A Taxonomy of Uncertainty Visualization Approaches". *Uncertainty Quantification in Scientific Computing*. Springer Berlin Heidelberg, 2012, 226–249. ISBN: 978-3-642-32677-6. DOI: 10.1007/978-3-642-32677-6_15 4.

[PRT*21] PUGMIRE, DAVID, ROSS, CAITLIN, THOMPSON, NICHOLAS, et al. "Fides: A General Purpose Data Model Library for Streaming Data". *High Performance Computing*. Springer International Publishing, 2021, 495–507. DOI: 10.1007/978-3-030-90539-2_34 8.

[PYK*18] PUGMIRE, DAVID, YENPURE, ABHISHEK, KIM, MARK, et al. "Performance-Portable Particle Advection with VTK-m". *Eurographics Symposium on Parallel Graphics and Visualization (EGPGV)*. 2018, 45–55. DOI: 10.2312/pgv.20181094 3.

[RV11] RAMACHANDRAN, PRABHU and VAROQUAUX, GA EL. "Mayavi: 3D Visualization of Scientific Data". *Computing in Science & Engineering* 13.2 (Mar. 2011), 40–51. DOI: 10.1109/MCSE.2011.35 2.

[SG24] SCHROEDER, WILL and GEVECI, BERK. *Happy Birthday VTK: 30 Years of Innovation*. Kitware Blog. Jan. 2024. URL: https://www.kitware.com/happy-birthday-vtk-30-years-of-innovation/ 2.

[SGA*22] STONE, JOHN E., GRIFFIN, KEVIN S., AMSTUTZ, JEFFERSON, et al. "ANARI: A 3-D Rendering API Standard". *Computing in Science & Engineering* 24.2 (2022), 7–18. DOI: 10.1109/MCSE.2022.3163151 4, 8.

[SK19] SULLIVAN, BANE and KASZYNSKI, ALEXANDER. "PyVista: 3D plotting and mesh analysis through a streamlined interface for the Visualization Toolkit (VTK)". *Journal of Open Source Software* 4.37 (May 2019), 1450. DOI: 10.21105/joss.01450 2.

[SLH*15] SEWELL, CHRISTOPHER, LO, LI-TA, HEITMANN, KATRIN, et al. "Utilizing many-core accelerators for halo and center finding within a cosmology simulation". *IEEE 5th Symposium on Large Data Analysis and Visualization (LDAV)*. 2015. DOI: 10.1109/LDAV.2015.7348076 3.

[SMG15] SCHROEDER, WILLIAM, MAYNARD, ROBERT, and GEVECI, BERK. "Flying edges: A high-performance scalable isocontouring algorithm". *Large Data Analysis and Visualization (LDAV)*. Oct. 2015. DOI: 10.1109/LDAV.2015.7348069 3.

[SML04] SCHROEDER, WILL, MARTIN, KEN, and LORENSEN, BILL. *The Visualization Toolkit: An Object Oriented Approach to 3D Graphics*. Fourth. ISBN 1-930934-19-X. Kitware Inc., 2004 1, 2, 4.

[SYB*21] SANE, SUDHANSHU, YENPURE, ABHISHEK, BUJACK, ROXANA, et al. "Scalable In Situ Computation of Lagrangian Representations via Local Flow Maps". *Eurographics Symposium on Parallel Graphics and Visualization (EGPGV)*. Winner: Best Paper. June 2021. DOI: 10.2312/pgv.20211040 3.

[TFL*18] TIERNY, JULIEN, FAVELIER, GUILLAUME, LEVINE, JOSHUA A., et al. "The Topology ToolKit". *IEEE Transactions on Visualization and Computer Graphics* 24.1 (Jan. 2018), 832–842. DOI: 10.1109/TVCG.2017.2743938 2.

[TSO*11] TURK, MATTHEW J., SMITH, BRITTON D., OISHI, JEFFREY S., et al. "yt: A Multi-code Analysis Toolkit for Astrophysical Simulation Data". *The Astrophysical Journal Supplement Series* 192.1 (Jan. 2011), 9. DOI: 10.1088/0067-0049/192/1/9 2.

[TSSM26a] TSALIKIS, SPIROS, SCHROEDER, WILLIAM, SZAFIR, DANIEL, and MORELAND, KENNETH. "An Accelerated Clip Algorithm for Unstructured Meshes: A Batch-Driven Approach using Data-Parallel Primitives". *Computer Graphics Forum* (June 2026). To appear in 3.

[TSSM26b] TSALIKIS, SPIROS, SCHROEDER, WILLIAM, SZAFIR, DANIEL, and MORELAND, KENNETH. "Memory-Aware External Facelist Calculation: A Data-Parallel Atomic Hash Counting Approach". *IEEE Transactions on Visualization and Computer Graphics* (Apr. 2026). To appear in 3.

[WAM*23] WANG, ZHE, ATHAWALE, TUSHAR M., MORELAND, KENNETH, et al. "$FunMC^2$: A Filter for Uncertainty Visualization of Marching Cubes on Multi-Core Devices". *Eurographics Symposium on Parallel Graphics and Visualization*. The Eurographics Association, 2023. ISBN: 978-3-03868-215-8. DOI: 10.2312/pgv.20231081 4.

[WHLS19] WANG, JUNPENG, HAZARIKA, SUBHASHIS, LI, CHENG, and SHEN, HAN-WEI. "Visualization and Visual Analysis of Ensemble Data: A Survey". *IEEE Transactions on Visualization and Computer Graphics* 25.9 (2019), 2853–2872. DOI: 10.1109/TVCG.2018.2853721 4.

[WHW16] WANG, RUONAN, HARRIS, CHRISTOPHER, and WICENEC, ANDREAS. "AdiosStMan: parallelizing casacore table data system using adaptive IO system". *Astronomy and Computing* 16 (July 2016), 146–154. DOI: 10.1016/j.ascom.2016.05.003 8.

[WJA*17] WALD, I, JOHNSON, GP, AMSTUTZ, J, et al. "OSPRay - A CPU Ray Tracing Framework for Scientific Visualization". *IEEE Transactions on Visualization and Computer Graphics* 23.1 (Jan. 2017), 931–940. DOI: 10.1109/TVCG.2016.2599041 8.

[WZA*24] WALD, INGO, ZELLMANN, STEFAN, AMSTUTZ, JEFFERSON, et al. "Standardized Data-Parallel Rendering Using ANARI". *2024 IEEE 14th Symposium on Large Data Analysis and Visualization (LDAV)*. Oct. 2024, 23–32. DOI: 10.1109/LDAV64567.2024.00013 8.

[YCM19] YENPURE, ABHISHEK, CHILDS, HANK, and MORELAND, KENNETH. "Efficient Point Merging Using Data Parallel Techniques". *Eurographics Symposium on Parallel Graphics and Visualization (EGPGV)*. 2019. DOI: 10.2312/pgv.20191112 3.

[ZAR21] ZHANG, MENGCHI, ALAWNEH, AHMAD, and ROGERS, TIMOTHY G. "Characterizing Massively Parallel Polymorphism". *2021 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. 2021, 205–216. DOI: 10.1109/ISPASS51385.2021.00037 5.

[ZS21] ZHENG, BOYAN and SADLO, FILIP. "Uncertainty in Continuous Scatterplots, Continuous Parallel Coordinates, and Fibers". *IEEE Transactions on Visualization and Computer Graphics* 27.2 (Feb. 2021), 1819–1828. DOI: 10.1109/TVCG.2020.3030466 4.

[ZSMG26] ZIMAN, ROXANNE, SAHARAN, SHEHRYAR, MCGILL, GAËL, and GARRISON, LAURA. ""It Looks Sexy but it's Wrong." Tensions in Creativity and Accuracy using genAI for Biomedical Visualization". *IEEE Transactions on Visualization and Computer Graphics* 32.1 (Jan. 2026), 320–330. DOI: 10.1109/TVCG.2025.3633883 8.