To appear in:
International Journal of High Performance Computing Applications
34(6):676–691
DOI: 10.1177/1094342020935991

A Terminology for In Situ Visualization and Analysis Systems

Hank Childs, Sean D. Ahern, James Ahrens, Andrew C. Bauer, Janine Bennett, E. Wes Bethel, Peer-Timo Bremer, Eric Brugger, Joseph Cottam, Matthieu Dorier, Soumya Dutta, Jean M. Favre, Thomas Fogal, Steffen Frey, Christoph Garth, Berk Geveci, William F. Godoy, Charles D. Hansen, Cyrus Harrison, Bernd Hentschel, Joseph Insley, Chris R. Johnson, Scott Klasky, Aaron Knoll, James Kress, Matthew Larsen, Jay Lofstead, Kwan-Liu Ma, Preeti Malakar, Jeremy Meredith, Kenneth Moreland, Paul Navrátil, Patrick O'Leary, Manish Parashar, Valerio Pascucci, John Patchett, Tom Peterka, Steve Petruzza, Norbert Podhorszki, David Pugmire, Michel Rasquin, Silvio Rizzi, David H. Rogers, Sudhanshu Sane, Franz Sauer, Robert Sisneros, Han-Wei Shen, Will Usher, Rhonda Vickery, Venkatram Vishwanath, Ingo Wald, Ruonan Wang, Gunther H. Weber, Brad Whitlock, Matthew Wolf, Hongfeng Yu, Sean B. Ziegeler

Abstract

The term "in situ processing" has evolved over the last decade to mean both a specific strategy for visualizing and analyzing data and an umbrella term for a processing paradigm. The resulting confusion makes it difficult for visualization and analysis scientists to communicate with each other and with their stakeholders. To address this problem, a group of over fifty experts convened with the goal of standardizing terminology. This paper summarizes their findings and proposes a new terminology for describing in situ systems. An important finding from this group was that in situ systems are best described via multiple, distinct axes: integration type, proximity, access, division of execution, operation controls, and output type. This paper discusses these axes, evaluates existing systems within the axes, and explores how currently used terms relate to the axes.

Keywords

In Situ Processing, Scientific Visualization

1 Introduction

For decades, the dominant paradigm for visualization and analysis has been "post hoc" processing. With post hoc processing, simulation codes save data to permanent storage (e.g., "spinning disk"), and visualization and analysis programs load this data after it is stored. Simulation codes typically store data iteratively, checkpointing the state of the simulation at a given time (a "time slice"), advancing for a while, saving their state again, and so on.

"In situ processing," meaning visualizing or analyzing data as it is generated, is an alternative processing paradigm to post hoc. Historically, a number of terms have been used to describe in situ processing and its many variant strategies. However, the community has yet to agree on a consistent taxonomy. Notably, over the last decade, the term "in situ" has been broadly used to 1) describe a specific strategy for processing data, and 2) as an umbrella term for the entire processing paradigm. This mixed use leads to confusion both for the visualization and analysis community and for its stakeholders.

The phrase "in situ" comes from Latin, and translates to "on site," "in position," or "in place." The term "in situ" very precisely describes the strategy where a visualization or analysis algorithm is applied to simulation data that has not been moved (i.e., is already in the processor's memory). However, the concept of "processing data as it is generated" applies much more broadly than to just data in registers, causing some researchers to question whether "in situ" is the best term to describe the overall processing paradigm. To illustrate this concern, consider the case where simulation data is moved to distinct resources for visualization, e.g., nodes dedicated for visualization and analysis on a supercomputer. On the one hand, the term "in situ" still applies, since the data is being processed "in place," i.e., resident on the same computer (or supercomputer). On the other hand, the data has been moved to distinct resources. In this scenario, is data still being processed "in place?"

While the term "in situ" is dominant today, early research used equally applicable terms, often in reference to specific variants: "concurrent processing" Ellsworth et al. (2006) to refer to processing data at the same time as the simulation is running, "co-processing" Haimes (1994); Haimes and

Corresponding author:
Hank Childs, University of Oregon, Eugene, Oregon, USA
Email: hank@uoregon.edu

Barth (1995); Haimes and Edwards (1997); Fabian et al. (2011); Ayachit et al. (2016a) to refer to visualization routines directly coupled with simulation code, and "runtime visualization" Ma (1996); Tu et al. (2006); Insley et al. (2007) to refer to applying visualization in place. In each case, the terms used previously were likely as suitable in describing this processing paradigm as the "in situ" term, although they did not ultimately garner the same popularity.

That said, the terminology problems we consider go well beyond whether or not "visualizing or analyzing data as it is generated" should be described using the term "in situ." Our community also uses a variety of sub-terms, like "in transit," "in-line," "loosely-coupled," and "tightly-coupled," to describe specific forms of in situ processing. These terms are not used consistently, creating confusion within the community, and this confusion served as the main motivation for our effort.

To remedy this lack of consistent terminology, a group of visualization and analysis practitioners convened over the course of a year; this paper summarizes the outcome of their efforts. An important contribution of this effort is the identification of six axes to more precisely characterize in situ systems. These axes show that there are a diverse set of approaches behind in situ processing. Another important contribution is our new proposed terminology for in situ systems. In our terminology, an in situ system is described via the options it employs for each of the six axes. As a further contribution, we analyze existing systems and terms within the axes.

The paper is organized as follows:

- Section 2 defines the six axes to describe an in situ system, as identified by our group.
- Section 3 describes some notional in situ systems, and classifies them according to our axes.
- Section 4 describes how to apply our axes in the context of complex workflows.
- Section 5 looks at recent in situ systems and classifies them based on our axes.
- Section 6 documents the process our group used to organize, discuss issues, and reach consensus.

Finally, our group also discussed whether to continue advocating for the usage of the term "in situ" to describe "processing data as it was generated." Ultimately, we decided that we were in favor of continuing to use the term, although consensus was not achieved on this point. In a vote, 70% of our participants supported continuing to use the "in situ" term, in large part because it had too much inertia to reverse course. In particular, it was noted that this term has been adopted by our stakeholders and funding agencies, and promoting an alternate term — even if more precise — could create confusion. On the other side, 30% of our participants voted that we should focus on a more appropriate term.

2 Axes of In Situ Systems

First, we use the phrase "in situ system" to describe an in situ software solution. Specifically, we use the word "system," since there are often multiple components to coordinate, whether across nodes on a supercomputer, across multiple in situ instances, or even between the in situ routines and the simulation code.

When choosing our axes, our group's focus was on better distinguishing between current in situ systems being lumped together as similar, although they were taking different approaches. To this end, our group identified six axes to describe an in situ visualization and analysis system:

- Integration Type: how visualization and analysis routines are integrated into the simulation code.
- Proximity: how close the visualization and analysis routines are to the data.
- Access: how the simulation makes data available to visualization and analysis routines.
- Division of Execution: how compute resources are shared between simulation and in situ routines.
- Operation Controls: the mechanism for selecting which operations are executed during run-time.
- Output Type: which types of operations are performed on the simulation data before it is output.

Figure 1 provides an overview of the terms used for each axis, and the remainder of this section describes these terms in more detail.

Using our terminology, a system is described by its choices for each of the six axes. Exemplar systems for common instantiations ("loosely-coupled," "tightly-coupled," etc.) are explored in Section 3.

2.1 Integration Type

Integration type refers to how the in situ visualization and analysis routines are integrated into the simulation code. In the majority of implementations, the simulation code is aware of the integration and makes calls in support of data marshaling. However, it is also possible to integrate in situ routines without the simulation being aware. We use this distinction — **Application-Aware** versus **Application-Unaware** — as the top-level category describing integration type.

We identified three distinct sub-categories of application-aware integrations, although these sub-categories may be viewed as points along a spectrum. The first, **Bespoke**, refers to the case where custom visualization and analysis routines are written specifically for a single simulation code, and is tailored to its needs. This is also sometimes referred to as "embedded routines." The latter two sub-categories of application-aware integrations cover configurations where systems are integrated into the simulation code, and data is marshaled into those frameworks via APIs. The second sub-category, **Dedicated API**, describes the case where the system is dedicated to visualization and analysis, and so the simulation code is aware that interactions with this API are for the purpose of visualization and analysis. This is the approach used by systems like VisIt/Libsim and ParaView/Catalyst. The final sub-category, **Multi-purpose API**, describes the case where the scope of the system is data, meaning that it includes visualization and analysis, but that it also might include I/O or data movement between components. This is the approach used by systems such as ADIOS. With multi-purpose API, the simulation code may or may not be aware whether the API is doing visualization

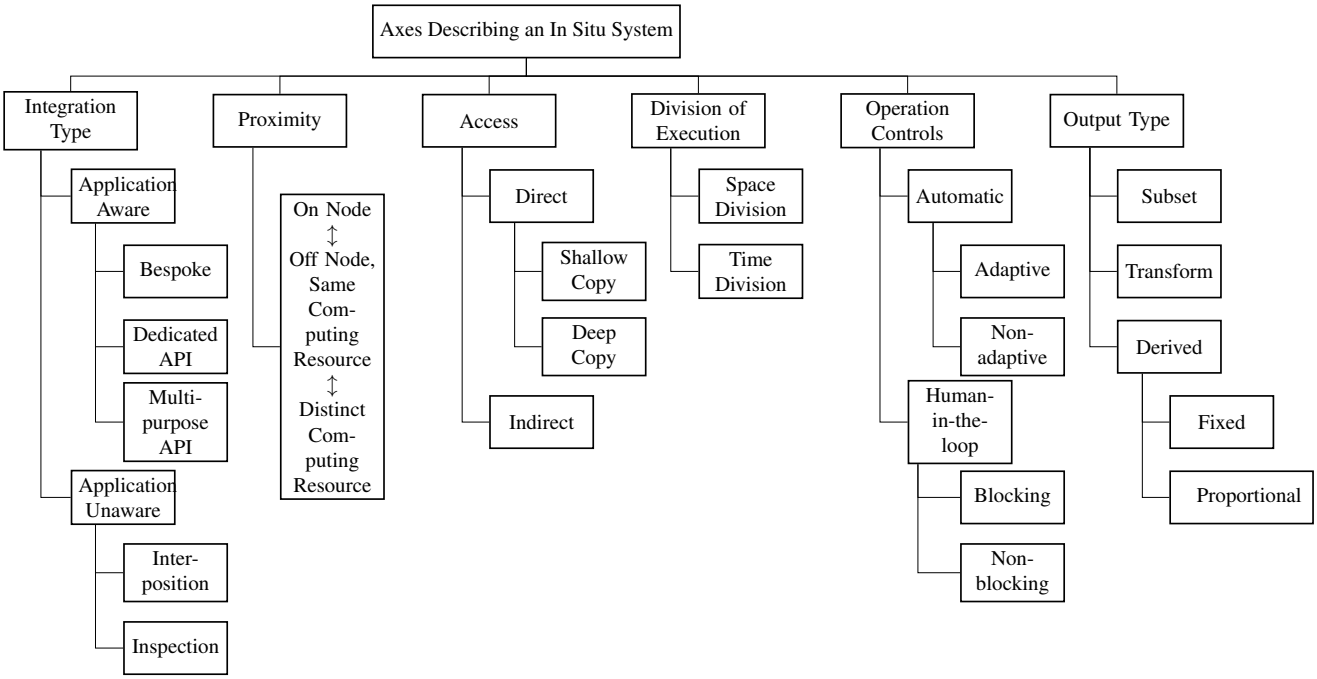

Figure 1. The top of this diagram has our six axes for describing in situ systems. Underneath each of the six axes are its corresponding categories and sub-categories.

and analysis tasks. We still refer to this case as Application-Aware, since the simulation code is aware of the framework's API, and does data marshaling to support the framework.

We identified two sub-categories of application-unaware integration types. **Interposition**, the first sub-category, refers to the practice of creating a dynamically-loaded library which contains symbols known to the simulation code, and inserting this library into the place of the original library that the simulation code was expecting. For example, if a simulation code writes data using the MPI-IO library, then an interposition approach would create a new library with function names matching those of MPI-IO, would have its implementations of those functions perform in situ processing, and would swap the new library in for the MPI-IO library at runtime. Note that, although both this example and the example for **Multi-purpose API**, focused on I/O, the key distinction for the **Interposition** approach is that a library interface is used for something other than its original intent. **Inspection**, the second sub-category, refers to the practice of inspecting memory to infer patterns in data layout and automatically adding in situ processing. Inspection-based in situ relies on system facilities used by tools such as debuggers and profilers. Finally, the application-unaware approach is relatively new for in situ processing, and new sub-categories may need to be added as this approach evolves.

There are three main considerations motivating the five categories of integration type. One is the effort to integrate the in situ routines into the simulation code (referred to here as "simulation code effort"). Another is the effort to develop the in situ system (referred to here as "in situ system effort"). The final consideration is the reusability of the in situ system across multiple simulation codes. These last two considerations are related, as increasing reusability likely increases in situ system effort. Bespoke approaches often require minimal simulation code effort (since they are tailored to the simulation code) and in situ system effort (since the approach often requires a trivial system), but its reusability is often highly limited. Dedicated API and Multi-purpose API require much more simulation code effort and in situ system effort, but often have higher reusability. The application-unaware categories may require the highest in situ system effort, but they require no simulation code effort (by definition), and the reuse possibilities are high.

2.2 Proximity

The Proximity axis characterizes the cost to access data. This cost could be in time (how fast can we access data?) or in energy (how much energy is required to access data?).

When considering proximity, it is important to consider the path from where the data resides to where it should be processed. That said, there are myriad possible configurations this path can take. As such, we view this axis as a continuous spectrum, not a discrete one with a fixed number of choices. This is particularly true given innovations in architecture, as any attempt to enumerate all options would likely become stale quickly.

We divide the spectrum of options for proximity into three broad categories:

- **On Node**
- **Off Node, Same Computing Resource**
- **Distinct Computing Resource**

With **On Node** access, the memory hierarchy forms the basic model. The closest access occurs when visualization or analysis algorithms are applied to data that is in the memory data registers, followed by options such as L1-cache, L2-cache, L3-cache, and random-access memory. Beyond this are options such as NUMA accesses to memory on other sockets, non-volatile memory on node, and local disks. Placement for each of these options onto a spectrum requires

understanding of latency and bandwidth, and may vary based on architecture, especially as hardware components improve over time (e.g., NVLink).

With **Off Node, Same Computing Resource**, there are fewer options: traveling one switch between nodes, two switches, etc. Of course, there still may be costs within a node, i.e., costs from pulling data from NVRAM on a node to send it over the network to another node, which then places it in an accelerator's memory. In these cases, all costs incurred along the path from where the data originally resides to location where it is processed should be considered.

While **Distinct Computing Resources** strains the usage of the term "in situ," it fits within our terminology since we define in situ processing as processing data as it is generated. As an example, consider when data is streamed (maybe in a reduced form) from a simulation's source to scientists in remote locations, who can then explore the data using local resources. We call this "in situ," since the data is being processed while it is generated, but clearly it is not "in place." Further, it is worth noting that a recent Department of Energy workshop on workflows Deelman et al. (2015) made a different decision, and decided this use case should not be called in situ processing.

2.3 Access

Access refers to how the simulation makes data available to visualization and analysis routines. We consider this axis from the perspective of logical memory spaces — by considering virtual addresses, the axis generalizes to a variety of memory models. The main options for access are **Direct** access (where the in situ routines share the same logical memory space as the simulation code) and **Indirect** access (where in situ routines run in a distinct logical memory space from the simulation code). Sometimes Access is conflated with Proximity, because **Direct Access** often occurs with **On Node Proximity** and **Indirect Access** often occurs with **Off Node Proximity**. However, these axes can pair oppositely. For example:

- **Direct Access** and **Off Node Proximity** pair when a simulation code exposes data via remote direct memory access (RDMA) or a partitioned global address space (PGAS).
- **Indirect Access** and **On Node Proximity** pair when the simulation code and in situ routines run as separate processes (to minimize integration effort) and exchanged data via the network or a local filesystem.

Within **Direct Access**, we distinguish between **Deep Copy** and **Shallow Copy** implementations. With **Deep Copy** implementations, in situ routines make a copy of their input data from the simulation. One reason for using **Deep Copy** is because the routines use a fixed data structure, and this data structure does not match the simulation code, for example because the simulation stores data in column-major order and the in situ routine assumes row-major order. This approach is expedient for software development, but suboptimal in terms of resources, specifically using extra memory and taking extra time to copy data. Another motivation for the **Deep Copy** approach is to prevent stalling the simulation — if the in situ routine makes a copy, then the simulation is free to proceed even if the in situ routine is not finished. **Shallow Copy** implementations, on the other hand, adapt their data structures to those of the simulation code. SCIRun Johnson et al. (2000) did this by using a templated approach and adapting to the simulation code at compile time. EAVL Meredith et al. (2012) did this through a data representation that contained other arrays, including arrays from simulation data, and then accessing data via a layer of indirection. VTK Schroeder et al. (2004) takes a similar approach to EAVL, although it handles variation in data layout via virtual functions.

Both **Direct Access** and **Indirect Access** must deal with data synchronization, as data should neither be consumed nor overwritten too early. For the **Direct Access** case, this can be achieved using standard synchronization methods, e.g., mutexes. For the **Indirect Access** case, this can be achieved via a communication protocol between simulation code and in situ system. There are several options for dealing with a simulation producing data faster than the in situ system can handle. These include: stalling the simulation until new data can be taken on; buffering raw data, which will, however, drive up memory consumption; and aborting the in situ routine and restarting it on the new data.

2.4 Division of Execution

Division of Execution refers to how compute resources are shared between simulation and in situ routines. The two categories within Division of Execution are:

- **Space Division**. The simulation and in situ routines are mapped to disjoint physical compute resources. That is, a subset of the compute resources is exclusively dedicated to in situ routines.
- **Time Division**. The simulation and in situ routines are both mapped to the same physical compute resources. Some (or all) of the compute resources alternate between advancing the simulation and visualization and analysis. That is, no compute resources are exclusive to in situ routines.

The execution time and memory usage required by in situ routines are generally less than those needed by the simulation, often by a significant amount. Regardless, division of execution between simulation code and in situ system is a critical issue for the efficiency of the overall system — allocating insufficient resources or insufficient duration to an in situ system can slow down the simulation code. That said, it is sometimes difficult to assess the necessary computational resources and duration for an in situ system to complete its tasks, since factors such as algorithm scalability, computational bottlenecks, and sensitivities to data layouts have large impacts on performance. Fortunately, the division need not be fixed, as the simulation can choose to adapt resource usage with many combinations of integration type, proximity, and access. Indeed, current research seeks to enable co-scheduling, i.e., dynamic management of system resources at the runtime-system-level Pebay et al. (2016); Peterson et al. (2015); Kale and Krishnan (1993), which removes the complexity of dynamic resource management from the application developer altogether.

Each division strategy has potential benefits and pitfalls, with manifestations varying across in situ configurations. **Space Division** facilitates both the efficient execution of the

simulation as well as the appropriation of an ideal set of resources to in situ routines. However, variations in the scales and runtimes of the routines could lead to under-utilized or oversubscribed subsets of resources. Managing this synchronization as well as possibly necessary data transfers may require significant additional infrastructure. **Time Division** requires substantially less (or no) synchronization and data transfer efforts. However, while in situ routines are frequently I/O bound in many instances, optimal efficiency is contingent upon data partitioning. For instance, the parallel scaling of visualization algorithms relies on infrastructure which can be sensitive to the size and shape of data domains, for example ghost data generation. The domain decomposition native to a simulation is sometimes unfavorable for analysis and visualization, an issue more easily addressable with post processing or **Space Division**. Both **Space Division** and **Time Division** can have significant cost if done poorly: **Space Division** can possibly block the simulation if there are not enough resources to keep up, while **Time Division**, when visualization or analysis routines run slowly, prevents the compute resources from being returned to the simulation.

2.5 Operation Controls

Operation Controls describes the mechanism for selecting which operations are executed during run-time. We identify two major categories within operation controls — **Automatic** and **Human-in-the-Loop** — both of which have sub-categories.

With **Automatic** Operation Controls, users select which operations to perform in advance of the calculation, and there is no human-in-the-loop during the simulation's execution. Within this category, we have identified two sub-categories. With the **Adaptive** sub-category, the in situ routines can adapt which operations are performed as the simulation executes. As an example, some key criteria may trigger the execution of some routines that were not executed otherwise. With the **Non-adaptive** sub-category, the in situ routines are static.

With **Human-in-the-Loop** Operation Controls, stakeholders modify which visualization and analysis routines are executed in situ. With the **Blocking** sub-category, the simulation can pause when waiting for guidance from a stakeholder. With the **Non-blocking** sub-category, the simulation will not pause to wait for input from a stakeholder.

2.6 Output Type

Output Type describes which operations the in situ system performs on the simulation data before it is output (meaning either stored or sent to another in situ sub-system). While a system can be described without understanding its output, our group felt that this was a worthy inclusion regardless — this category speaks to output size, which is an important consideration. We identify three major categories for output type: **Subset**, **Transform**, and **Derived**.

Subset refers to operations where a subset of the data is selected, and the rest is discarded. Examples include subsampling (e.g., coarse versions of the data), focusing on regions of interest, or extracting portions with a certain property, as with query-driven visualization Stockinger et al. (2005) or as with topologies queries Heine et al. (2016) (e.g., N largest connected components).

Transform refers to operations that are performed on each element of the data. Our notion of **Transform** does not include reduction, meaning that we expect the data sets created by the transformation process are the same scale as the input data. Wavelet transformations would be an example of a transform that may be applied in situ.

Derived refers to operations that generate new data of a different nature than the input. Within the **Derived** category, we consider two sub-types: **Fixed** and **Proportional**. Products of **Fixed** operations are independent of the input size. Examples include statistical summarizations, like with Ye et al. (2016), and rendered images (when the image size is fixed). Products of **Proportional** operations vary based on input size. Examples include isosurfaces, indexing, intermediate visualization representations, and topological analysis. **Proportional** operations imply that the amount of data saved varies with the input data size, but some operations' proportion are data dependent while others are not. For example, the amount of data to save when isosurfacing depends on both the input data and the isovalue, but the amount of data to save when subsampling is not data dependent.

Some operations can be used in either **Fixed** or **Proportional** approaches. For example, Lagrangian basis flow extraction Agranovsky et al. (2014) can output a fixed size (and potentially miss information about the vector field) or a proportional size (and thus be more likely to capture information about the vector field).

Finally, the value for Output Type for an in situ routine can be more than a single entry. For example, wavelet compression can be accomplished by first doing a wavelet transform, and then discarding the least important coefficients. This would be categorized as **Transform — Subset**, which indicated that the data is transformed before being reduced by a subset operation. Finally, some large, dedicated in situ systems offer many simultaneous output types, and may need multiple descriptions to describe those outputs.

3 Classifying Notional In Situ System Examples

In this section, we describe three notional systems, and classify them according to our axes. Also, note that the terms used in the subsection headings (tightly-coupled, loosely-coupled, hybrid in situ) are ambiguous and can have multiple interpretations, although we believe the examples specified fall within most accepted definitions.

3.1 Example 1: Tightly-Coupled System

Consider the following system: A simulation code links an in situ library into its code. When the simulation code calls a function in the in situ API, it both specifies the operations to perform and sends data to operate on. The simulation code's usage of the API is static; the simulation code compiles against the API, and the same function is called at a regular interval. When the simulation code invokes the in situ function, the in situ library immediately executes its operations on the same hardware, first transforming it to its

own data model, then applying the specified operations, and finally creating images that are saved to disk. The function then returns and the simulation code resumes execution.

In our terminology, this would be classified as:

- Integration Type: Dedicated API
- Proximity: On Node
- Access: Direct: Deep Copy
- Division of Execution: Time Division
- Operation Controls: Automatic: Non-adaptive
- Output Type: Derived: Fixed

3.2 Example 2: Loosely-Coupled System

Consider the following system: A simulation code links in an API for data management. When the simulation code calls functions in the in situ API, it believes it is doing I/O operations. However, the in situ library instead sends data to remote nodes which are dedicated to visualization and analysis. A user is running a visualization and analysis tool on these remote nodes, interacting with the data as it comes over the network. When a new time slice comes over the network, the data the user was looking at is flushed and replaced with the new data.

In our terminology, this would be classified as:

- Integration Type: Multi-purpose API
- Proximity: Off Node
- Access: Indirect
- Division of Execution: Space Division
- Operation Controls: Human-in-the-loop: Non-blocking
- Output Type: Derived: Fixed

3.3 Example 3: Hybrid In Situ System

Consider the following system whose sole purpose is to render isosurfaces. In this system, the desired isovalues result in a sparse isosurface (i.e., few triangles compared to the number of cells), so, when data is produced, an isosurfacing routine is immediately applied. This routine was written specifically to work on data from this simulation code. The resulting triangles are sent over the network to dedicated visualization nodes, using a data transfer library. There, separate visualization software renders the data. Since the location of the isosurface varies, the software evaluates the data set and determines the best camera angles to capture the data. It saves the resulting images to disk.

In this case, this system is actually two distinct sub-systems operating in tandem. We classify the sub-systems separately. More discussion of this topic is in the next section (In Situ Workflows).

- Sub-system #1
  - Integration Type: Bespoke
  - Proximity: On Node
  - Access: Direct: Shallow Copy
  - Division of Execution: Time Division
  - Operation Controls: Automatic: Non-adaptive
  - Output Type: Derived: Proportional
- Sub-system #2
  - Integration Type: Multi-purpose API
  - Proximity: Off Node
  - Access: Indirect
  - Division of Execution: Space Division
  - Operation Controls: Automatic: Adaptive
  - Output Type: Derived: Fixed

4 In Situ Workflows

In situ systems sometimes operate in a form where there are multiple, distinct sub-systems, which operate in a workflow-like fashion. That is, sub-system "A" will transform data and transport it to sub-system "B," sub-system "B" will transform data and transport it to sub-system "C," and so on. Of course, the flow of data does not need to be sequential from "A" to "B" to "C," but instead can flow in arbitrary ways, including forming cycles, acting as a source for multiple sub-systems, accepting input from multiple sources, etc. In some cases, "A," "B," etc., are the same program, but this program was invoked in a way that causes it to function differently. In other cases, the sub-systems are distinct programs, but those programs come from the same source code repository, and are branded under the same product name. In still other cases, the sub-systems are truly distinct pieces of software.

For our categorization, we classify each sub-system in the workflow separately (as seen in Example 3). That said, many workflows contain sub-systems that do not relate to visualization and analysis; when classifying an in situ system, we recommend only including sub-systems that do visualization and analysis operations in a categorization. Finally, note that if the sub-systems have non-sequential flow (i.e., splitting output, cycles, etc.), then the classifications listing for each sub-system would need to be augmented with a graph that captures the respective inputs and outputs of each sub-system.

5 Surveying Existing In Situ Systems

In this section, we consider 15 existing in situ systems. The systems we surveyed were the ones suggested by our co-authors; we believe the resulting list is representative, but not exhaustive. Table 1 summarizes these in situ systems based on our axes to describe an in situ system.

We divide our treatment into three sections. Seven of the fifteen systems do exclusively **Time Division** with **On Node** proximity; these are discussed in Section 5.1. Six of the fifteen systems allow for choice with respect to Division of Execution (i.e., both **Time Division** and **Space Division**) and Proximity (i.e., ranging from **On Node** to **Off Node**, with a few also supporting **Distinct Computing Resources**); these are discussed in Section 5.2. Finally, the remaining two systems made specific choices for Division of Execution and Proximity that were different than those of Section 5.1; these are discussed in Section 5.3.

5.1 *Time Division and On Node Proximity*

The seven systems in this section all use **Time Division** and **On Node** proximity. This means they all use a model where the simulation advances, then pauses and hands control to the in situ system which completes its operations, hands control back, and so on. The seven systems are:

Ascent, Freeprocessing, ParaView/Catalyst, SCIRun, QIso, VisIt/Libsim, and XImage.

Four of the systems (ParaView/Catalyst, SCIRun, VisIt/Libsim, and XImage) use **Dedicated API** — when simulation codes connect to these systems, it is known to be for the purpose of visualization and analysis tasks. Some other noteworthy aspects of these in situ systems:

- Libsim Whitlock et al. (2011) and Catalyst Bauer et al. (2015); Fabian et al. (2011) are in situ libraries that deliver the capabilities of VisIt Childs et al. (2012) and ParaView Ayachit (2015), respectively. In both cases, a post hoc tool began around the year 2000, became popular, and was then reworked to have an in situ form during the ensuing decade. Both libraries support shallow-copying of simulation data arrays that conform to the contiguous memory layout used natively by the VTK library. For other memory layouts, shallow copy support is possible via subclassing of VTK's data array modules. In both cases, users can provide scripts prior to runtime that control visualization and analysis (**Automatic: Non-Adaptive**) or allow users to control the visualization directly (**Human-in-the-loop: Blocking**). Finally, in conjunction with other technologies, such as ADIOS, these tools can be used to perform (**Human-in-the-loop: Non-blocking**) by using separate, dedicated visualization resources.
- SCIRun Parker and Johnson (1995); Knežević et al. (2012); Parker et al. (1997b,a) is different than Catalyst and Libsim — it is not delivering a post hoc tool, but rather was designed with in situ in mind from the beginning. Overall, SCIRun is a scientific programming environment designed for interactive construction, debugging, and steering of large-scale scientific computations. It is a framework in which large scale computer simulations can be composed, executed, controlled and tuned interactively. It depends on a data flow design where users can design and modify simulations interactively via a dataflow programming model, and this design allows for easy extensibility of new modules. An especially interesting design choice for SCIRun was the decision to incorporate templates with its data model. This allowed SCIRun to adapt its data model at compile-time to minimize memory footprint, achieving **Shallow Copy** at a higher rate than other systems.
- XImage Ye et al. (2018) is an in situ library committed to **Fixed** output type. It produces only images, although these images are "explorable," meaning that each produced image is a meta-image that can produce additional images Tikhonova et al. (2010a). It currently supports volume rendering Tikhonova et al. (2010b) and pathtube visualization Ye et al. (2013).

The remaining three systems have different choices for integration type. They are:

- Ascent Larsen et al. (2017) uses a **Multi-purpose API**, and simulation codes can use Ascent to perform their I/O, as well as visualization and analysis. Ascent uses VTK-m Moreland et al. (2016), which adapts its data model to match simulation code layout, enabling **Shallow Copy** in many instances. Ascent also has invested in **Automatic: Adaptive** workflows, and has an extensive system for triggers, which can be customized by users Larsen et al. (2018). Finally, Ascent can incorporate Jupyter notebooks to achieve **Human-in-the-loop: Blocking** Ibrahim et al. (2019).
- Freeprocessing Fogal et al. (2014) is one of the few systems we consider that uses **Interposition**. Its focus has been on ease of use and programmability, and works via a symbiont that uses binary instrumentation. The data that it extracts can then be forwarded to custom visualization routines, or to existing visualization tools (which would share the same resources).
- QIso Ziegeler et al. (2015) is one of only two **Bespoke** examples we consider. The library is dedicated to generating and rendering isosurfaces in MPI-based simulations. It uses the Marching Cubes algorithm Lorensen and Cline (1987) on structured grids obtained directly from a simulation's arrays, renders to an off-screen framebuffer, and parallel-composites the result via MPI to a single image. A library like this is powerful since it has minimal dependencies and is simple to incorporate.

5.2 Choice in Division of Execution and Proximity

The six systems in this section — ADIOS, Damaris, DataSpaces, GLEAN, libIS, and SENSEI — all provide options with respect to Division of Execution and Proximity. In particular, each of these systems can execute like those from Section 5.1. However, they can also operate in other ways, for example running visualization/analysis alongside the simulation on the same node, or sending data to distinct nodes. A popular **Off Node** form is "data staging" Oldfield et al. (1998), where data is moved from the simulation nodes to a smaller pool of visualization/analysis/I/O nodes, where data can be aggregated, processed, indexed, and filtered. This sometimes has cost benefits for in situ visualization and analysis Kress et al. (2019), but is regularly helpful for I/O, since dramatic reductions can be achieved in the total data volume to store.

ADIOS and GLEAN both use **Multi-Purpose APIs**, although GLEAN also can use an **Interposition** approach. Some other noteworthy aspects of these systems:

- ADIOS, short for the Adaptable I/O System (ADIOS) Liu et al. (2014), has a strong focus on I/O (in addition to visualization and analysis), and has been able to demonstrate strong I/O throughput for simulations. The ADIOS API allows writing to storage arrays and integrating with other workflow or data analytics systems without detailed knowledge of the underlying software and hardware stack (**Multi-Purpose API**). This API allows users to combine data storage, data staging, data compression, and/or data reduction. Noteworthy products integrated into ADIOS include ZFP Lindstrom (2014), SZ Di and Cappello (2016) and BZip2 for data compression

and I/O, ICEE Choi et al. (2013), FlexPath Dayal et al. (2014), and DataSpaces Docan et al. (2012) for data staging, and ParaView, VisIt, and Ascent for visualization and analysis.

- GLEAN Vishwanath et al. (2011) has a focus on taking application, analysis, and system characteristics into account to facilitate simulation-time data analysis and I/O acceleration. In other words, its focus is on being able to provide the least cost to carry out an operation. Another important focus for GLEAN is to provide an interface for in situ analysis with zero or minimal modifications to the existing application code base. It achieves non-intrusive integration by embedding in higher-level I/O libraries such as PnetCDF and HDF5 (**Interposition**).

Damaris and SENSEI both use **Dedicated APIs**, and both have the potential to share the same memory space with the simulation (i.e., **Direct** access in addition to **Indirect** access):

- Damaris Dorier et al. (2016) began as middleware for I/O operations, and has increasingly added support for visualization, first with VisIt, then with Catalyst. Damaris operates with an XML-based system, which was first used to allow users to describe the data semantics to the backend I/O libraries, and subsequently used to control in situ visualization with VisIt. Finally, Damaris started with a focus on **On Node**, and then added an **Off Node** option after its initial deployment.
- SENSEI Ayachit et al. (2016b) provides a generic API for in situ processing in order to enable a "write-once, run-anywhere" environment. One advantage from their approach is proximity portability, i.e., runtime selection between running **On Node** or **Off Node**. Another advantage is tool portability, i.e., runtime selection between in situ technologies, such as Libsim, Catalyst, and Ascent, as well as by enabling use of custom user-written methods, such as parallel Python scripts. Further, SENSEI enable interoperability between the tools they integrate.

The OSPRay libIS library Usher et al. (2018) can do both **Space Division** and **Time Division**, but it focuses solely on **Indirect** access. libIS exposes a **Dedicated API** to the simulation, which it uses to listen for visualization clients, and send data to these clients. Clients can connect and disconnect to the simulation as needed, and, once connected, request to receive data for the current time step.

Finally, while data staging was mentioned at the beginning of this section and again in the ADIOS section, we briefly focus on one data staging technology, DataSpaces Docan et al. (2012), as an example system. DataSpaces is a programming system that provides data exchange services to support extreme-scale in situ workflows. It can be accessed by many components and services in a workflow; components/services can dynamically connect to it and use it to coordinate their execution and to support dynamic and asynchronous interactions and data exchange among them. It also contains features similar to some of the systems above — data querying, filtering, and data redistribution. DataSpaces is built on an autonomic data-management layer that leverages machine learning algorithms and application hints to support application/system-aware data placement and movement Subedi et al. (2018); Jin et al. (2015), and an asynchronous, low-overhead, memory-to-memory data transport substrate based on RDMA one-sided communications.

5.3 Remaining Systems

The commercial ANSYS EnSight tool Frank and Krogh (2012) focuses on **Space Division**. Its goal is to allow users to explore data from a currently running simulation. EnSight performs in situ analysis where new data from a running simulation is continuously discovered and loaded. They refer to this capability as "simulation monitoring." The capability is implemented via the file system — when new files are found, EnSight loads the new time steps into its set of time steps and updates its display. That said, the user can remain on the current time step or automatically jump to the latest one discovered. Although this usage pattern is performed via files on disk, we include it in the survey, since it fits our definition of "processing data as it is generated."

InSt Malakar et al. (2010, 2011) is the other **Bespoke** effort we consider, as it is strictly for weather applications. InSt works in two settings. First, InSt automatically detects critical weather events, such as cyclones, and refines the simulation, to ensure that the simulation progresses without stalling even with application dynamics. This is similar to SCIRun in terms of a computational steering component. Second, InSt also enables online remote visualization at the user's site, from where the user can concurrently visualize the simulation output as well as steer the simulation. This is an instance of **Distinct** resources, as domain scientists around the world can visualize their data as the simulation as running, and also adapt those visualizations.

6 Process for In Situ Terminology Project

This section describes the process for generating our terminology. There were four main phases to this effort: (1) form a community, (2) agree on the main axes of terminology, (3) agree on the options for each axis, and (4) document the result.

The main method of forming a community was via outreach at conferences. An open call for participation was made at a panel at the IEEE Symposium for Large Data Analysis and Visualization (LDAV) and also at a lightning talk at the Supercomputing Workshop on In Situ Analysis and Visualization (ISAV). This led to approximately thirty participants. Following this step, invitations were sent out to senior voices in the community, almost all of which were accepted.

Agreeing on the main axes of terminology and the options for each axis was primarily done by teleconference. In total, twelve conference calls were convened, featuring twenty to thirty participants each. The early teleconferences were devoted to deciding on the main concepts. This includes a deep discussion of the scope of the term "in situ" itself, and establishing that the terminology would focus on classifying a system by stating its choices for largely orthogonal axes, before specifying the axes themselves.

Table 1. Categorization of various existing in situ systems with respect to terminology. Table entries are ordered alphabetically.

Name	Ref	Integration Type	Proximity	Access	Division of Execution	Operation Controls
ADIOS	Liu et al. (2014)	MP-API	On Node; Off Node; Distinct	D-SC*; Indirect	Time; Space	Automatic: Adaptive; Automatic: Non-adaptive; Human-in-the-loop: Non-Blocking
ANSYS EnSight	Frank and Krogh (2012)	Interposition	Off Node; Distinct	Indirect	Space	Human-in-the-loop: Non-Blocking
Ascent	Larsen et al. (2017)	MP-API	On Node	D-SC; D-DC	Time	Automatic: Non-adaptive; Human-in-the-loop: Blocking
Damaris	Dorier et al. (2016)	D-API	On Node; Off Node	D-SC; D-DC; Indirect	Time; Space	Automatic: Non-adaptive; Human-in-the-loop: Blocking; Human-in-the-loop: Non-Blocking
DataSpaces	Docan et al. (2012)	MP-API	On Node; Off Node	D-DC; Indirect	Time; Space	Automatic: Adaptive
Freeprocessing	Fogal et al. (2014)	Interposition	On Node	D-SC	Time	Automatic: Non-adaptive
GLEAN	Vishwanath et al. (2011)	MP-API; Interposition	On Node; Off Node; Distinct	D-SC; D-DC; Indirect	Time; Space	Automatic: Adaptive; Automatic: Non-adaptive
InSt	Malakar et al. (2011)	Bespoke	On Node; Distinct	D-SC; Indirect	Time	Automatic: Non-adaptive; Human-in-the-loop: Non-Blocking
OSPRay LibIS	Usher et al. (2018)	D-API	On Node; Off Node	Indirect	Time; Space	Automatic: Adaptive; Automatic: Non-adaptive; Human-in-the-loop: Non-Blocking
ParaView	Ayachit (2015)	D-API	On Node	D-SC*	Time	Automatic: Non-adaptive; Human-in-the-loop: Blocking; Human-in-the-loop: Non-Blocking
QIso	Ziegeler et al. (2015)	Bespoke	On Node	D-SC	Time	Automatic: Non-adaptive
SENSEI	Ayachit et al. (2016b)	D-API	On Node; Off Node	D-SC*; Indirect	Time; Space	Automatic: Adaptive; Automatic: Non-adaptive; Human-in-the-loop: Non-Blocking
SCIRun	Parker and Johnson (1995)	D-API	On Node	D-SC; D-DC	Time	Automatic: Non-adaptive; Human-in-the-loop: Blocking; Human-in-the-loop: Non-Blocking
VisIt	Childs et al. (2012)	D-API	On Node	D-SC*	Time	Automatic: Non-adaptive; Human-in-the-loop: Non-Blocking
XImage	Ye et al. (2018)	D-API	On Node	D-SC	Time	Automatic: Non-adaptive; Human-in-the-loop: Non-Blocking

D-API = Dedicated API; MP-API = Multi-Purpose API
D-SC = Direct Integration - Shallow Copy; D-DC = Direct Integration - Deep Copy
* = does shallow copy when possible, otherwise deep copy

Subsequent teleconferences focused on specific options for each of the axes. Discussions revolved around both ideas and appropriate terminology – i.e., wording – that would convey those ideas.

Documenting the results was again done by group. Major discussion results were documented during each conference call. The lead author would review and refine these points after the call adjourned, which yielded an initial draft of the concepts discussed. This draft was then placed on Google Drive, and each co-author reviewed the document and commented on it. Subsequent teleconference calls then explored the comments in order to resolve them. Accepted comments were addressed by assigning a lead contributor to

fix the respective issue, and additional co-authors to review the change. Finally, all authors reviewed the final paper draft.

7 Conclusion

In this paper, we have presented a holistic terminology for the design space of in situ systems. Our proposed model comprises six axes along which an in situ system can be characterized; each of these axes has multiple sub-options. The motivation for this terminology is twofold. First, it serves as a starting point for a uniformly understood vocabulary that helps to navigate the in situ design space at large. Second, and perhaps more important, it helps characterize existing and possible systems, which facilitates identifying similarities and differences, and even possibly suggests new directions.

Developing this terminology by a community-driven approach is a key contribution towards the second aspect. The diverse discussions brought in many different perspectives in the beginning. Through a series of subsequently refined drafts, these were eventually distilled into the terminology described here. This process not only ensured that diverse views have been taken into account, but also provided the end result with a solid foundation and the backing of a significant number of practitioners in this community.

Finally, this terminology should not be considered static. As new systems are developed, we encourage future researchers to introduce new options and axes as appropriate. In response, we plan to update the terminology occasionally.

References

Agranovsky A, Camp D, Garth C, Bethel EW, Joy KI and Childs H (2014) Improved Post Hoc Flow Analysis Via Lagrangian Representations. In: *Proceedings of the IEEE Symposium on Large Data Visualization and Analysis (LDAV)*. Paris, France, pp. 67–75.

Ayachit U (2015) *The ParaView Guide: A Parallel Visualization Application*. Kitware. ISBN 978-1930934306.

Ayachit U, Bauer A, Duque EPN, Eisenhauer G, Ferrier N, Gu J, Jansen KE, Loring B, Lukić Z, Menon S, Morozov D, O'Leary P, Ranjan R, Rasquin M, Stone CP, Vishwanath V, Weber GH, Whitlock B, Wolf M, Wu KJ and Bethel EW (2016a) Performance analysis, design considerations, and applications of extreme-scale in situ infrastructures. In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC16)*. pp. 79:1–79:12.

Ayachit U, Whitlock B, Wolf M, Loring B, Geveci B, Lonie D and Bethel E (2016b) The SENSEI Generic In Situ Interface. In: *Proceedings of the Workshop on In Situ Infrastructures for Enabling Extreme-scale Analysis and Visualization (ISAV)*. IEEE Press, pp. 40–44.

Bauer AC, Geveci B and Schroeder W (2015) *The ParaView Catalyst User's Guide v2.0*. Kitware, Inc.

Childs H, Brugger E, Whitlock B, Meredith J, Ahern S, Pugmire D, Biagas K, Miller M, Harrison C, Weber GH, Krishnan H, Fogal T, Sanderson A, Garth C, Bethel EW, Camp D, Rübel O, Durant M, Favre JM and Navrátil P (2012) VisIt: An End-User Tool For Visualizing and Analyzing Very Large Data. In: *High Performance Visualization–Enabling Extreme-Scale Scientific Insight*. pp. 357–372.

Choi JY, Wu K, Wu JC, Sim A, Liu QG, Wolf M, Chang C and Klasky S (2013) Icee: Wide-area in transit data processing framework for near real-time scientific applications. In: *Workshop on Petascale (Big) Data Analytics: Challenges and Opportunities, held in conjunction with SC13*.

Dayal J, Bratcher D, Eisenhauer G, Schwan K, Wolf M, Zhang X, Abbasi H, Klasky S and Podhorszki N (2014) Flexpath: Type-Based Publish/Subscribe System for Large-scale Science Analytics. In: *IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*. pp. 245–255.

Deelman E, Peterka T et al. (2015) The Future of Scientific Workflows. Technical report, Report of the DOE NFNS/CS Scientific Workflows Workshop.

Di S and Cappello F (2016) Fast error-bounded lossy hpc data compression with sz. In: *IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. pp. 730–739.

Docan C, Parashar M and Klasky S (2012) Dataspaces: an interaction and coordination framework for coupled simulation workflows. *Cluster Computing* 15(2): 163–181.

Dorier M, Antoniu G, Cappello F, Snir M, Sisneros R, Yildiz O, Ibrahim S, Peterka T and Orf L (2016) Damaris: Addressing Performance Variability in Data Management for Post-Petascale Simulations. *ACM Transactions on Parallel Computing (ToPC)* 3(3): 15:1–15:43.

Ellsworth D, Green B, Henze C, Moran P and Sandstrom T (2006) Concurrent Visualization in a Production Supercomputing Environment. *IEEE Transactions on Visualization and Computer Graphics* 12(5): 997–1004.

Fabian N, Moreland K, Thompson D, Bauer AC, Marion P, Geveci B, Rasquin M and Jansen KE (2011) The paraview coprocessing library: A scalable, general purpose in situ visualization library. In: *IEEE Symposium on Large Data Analysis and Visualization (LDAV)*. pp. 89–96.

Fogal T, Proch F, Schiewe A, Hasemann O, Kempf A and Krüger J (2014) Freeprocessing: Transparent in situ visualization via data interception. In: *Eurographics Symposium on Parallel Graphics and Visualization*. pp. 49–56.

Frank R and Krogh MF (2012) The EnSight Visualization Application. In: *High Performance Visualization-Enabling Extreme-Scale Scientific Insight*. pp. 429–442.

Haimes R (1994) pv3-a distributed system for large-scale unsteady cfd visualization. In: *32nd Aerospace Sciences Meeting and Exhibit*. p. 321.

Haimes R and Barth T (1995) Application of the pV3 Co-Processing Visualization Environment to 3-D Unstructured Mesh Calculations on the IBM SP2 Parallel Computer. In: *Proc. CAS Workshop*.

Haimes R and Edwards DE (1997) Visualization in a parallel processing environment. In: *35th Aerospace Sciences Meeting and Exhibit*. p. 348.

Heine C, Leitte H, Hlawitschka M, Iuricich F, De Floriani L, Scheuermann G, Hagen H and Garth C (2016) A survey of topology-based methods in visualization. *Computer Graphics Forum* 35(3): 643–667.

Ibrahim S, Stitt T, Larsen M and Harrison C (2019) Interactive In Situ Visualization and Analysis using Ascent and Jupyter. In: *Proceedings of the In Situ Infrastructures on Enabling Extreme-Scale Analysis and Visualization (ISAV)*. pp. 44–48.

Insley JA, Papka ME, Dong S, Karniadakis G and Karonis NT (2007) Runtime visualization of the human arterial tree. *IEEE Transactions on Visualization and Computer Graphics* 13(4): 810–821.

Jin T, Zhang F, Sun Q, Bui H, Romanus M, Podhorszki N, Klasky S, Kolla H, Chen J, Hager R et al. (2015) Exploring data staging across deep memory hierarchies for coupled data intensive simulation workflows. In: *Proceedings of the IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. pp. 1033–1042.

Johnson C, Parker S and Weinstein D (2000) Large-Scale Computational Science Applications Using the SCIRun Problem Solving Environment. In: *Proceedings of the 2000 ACM/IEEE conference on Supercomputing*. URL http://www.sci.utah.edu/publications/crj00/super00_final.pdf.

Kale LV and Krishnan S (1993) Charm++: A Portable Concurrent Object Oriented System Based On C++. In: *Proceedings of the eighth annual conference on Object-oriented programming systems, languages, and applications*. pp. 91–108.

Knežević J, Mundani RP, Rank E, Khan A and Johnson CR (2012) Extending the SCIRun Problem Solving Environment to Large-Scale Applications. In: *Proc. of The IADIS Applied Computing 2012*.

Kress J, Larsen M, Choi J, Kim M, Wolf M, Podhorszki N, Klasky S, Childs H and Pugmire D (2019) Comparing the Efficiency of In Situ Visualization Paradigms at Scale. In: *ISC High Performance*. Frankfurt, Germany, pp. 99–117.

Larsen M, Ahrens J, Ayachit U, Brugger E, Childs H, Geveci B and Harrison C (2017) The ALPINE In Situ Infrastructure: Ascending from the Ashes of Strawman. In: *Proceedings of the In Situ Infrastructures on Enabling Extreme-Scale Analysis and Visualization (ISAV)*. pp. 42–46.

Larsen M, Woods A, Marsaglia N, Biswas A, Dutta S, Harrison C and Childs H (2018) A Flexible System for In Situ Triggers. In: *Proceedings of the Workshop on In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization (ISAV)*. pp. 1–6.

Lindstrom P (2014) Fixed-rate compressed floating-point arrays. *IEEE Transactions on Visualization and Computer Graphics* 20(12): 2674–2683.

Liu Q, Logan J, Tian Y, Abbasi H, Podhorszki N, Choi JY, Klasky S, Tchoua R, Lofstead J, Oldfield R et al. (2014) Hello ADIOS: the challenges and lessons of developing leadership class I/O frameworks. *Concurrency and Computation: Practice and Experience* 26(7): 1453–1473.

Lorensen WE and Cline HE (1987) Marching cubes: A high resolution 3d surface construction algorithm. *ACM siggraph computer graphics* 21(4): 163–169.

Ma KL (1996) Runtime volume visualization for parallel CFD. In: *Parallel Computational Fluid Dynamics*. Elsevier, pp. 307–314.

Malakar P, Natarajan V and Vadhiyar SS (2010) An adaptive framework for simulation and online remote visualization of critical climate applications in resource-constrained environments. In: *Conference on High Performance Computing Networking, Storage and Analysis (SC)*.

Malakar P, Natarajan V and Vadhiyar SS (2011) Inst: An integrated steering framework for critical weather applications. In: *Proceedings of the International Conference on Computational Science, ICCS 2011*. pp. 116–125.

Meredith JS, Ahern S, Pugmire D and Sisneros R (2012) EAVL: the extreme-scale analysis and visualization library. In: *Eurographics Symposium on Parallel Graphics and Visualization*. pp. 21–30.

Moreland K, Sewell C, Usher W, Lo L, Meredith J, Pugmire D, Kress J, Schroots H, Ma KL, Childs H, Larsen M, Chen CM, Maynard R and Geveci B (2016) VTK-m: Accelerating the Visualization Toolkit for Massively Threaded Architectures. *IEEE Computer Graphics and Applications (CG&A)* 36(3): 48–58.

Oldfield RA, Womble DE and Ober CC (1998) Efficient parallel I/O in seismic imaging. *The International Journal of High Performance Computing Applications* 12(3): 333–344.

Parker S, Beazley D and Johnson C (1997a) Computational steering software systems and strategies. *IEEE Computational Science and Engineering* 4(4): 50–59.

Parker S, Weinstein D and Johnson C (1997b) The SCIRun computational steering software system. In: *Modern Software Tools in Scientific Computing*. Boston: Birkhauser Press, pp. 1–40.

Parker SG and Johnson CR (1995) SCIRun: A Scientific Programming Environment for Computational Steering. In: *Supercomputing '95: Proceedings of the 1995 ACM/IEEE Conference on Supercomputing*. p. 52. URL https://ieeexplore.ieee.org/document/1383188.

Pebay P, Bennett JC, Hollman D, Treichler S, McCormick PS, Sweeney CM, Kolla H and Aiken A (2016) Towards asynchronous many-task in situ data analysis using legion. *IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)* : 1033–1037.

Peterson B, Dasari HK, Humphrey A, Sutherland J, Saad T and Berzins M (2015) Reducing overhead in the Uintah framework to support short-lived tasks on GPU-heterogeneous architectures. In: *International Workshop on Domain-Specific Languages and High-Level Frameworks for High Performance Computing (WOLFHPC'15)*. pp. 4:1–4:8.

Schroeder W, Martin K and Lorensen B (2004) *The Visualization Toolkit: An Object Oriented Approach to 3D Graphics*. Fourth edition. Kitware, Inc. ISBN 1-930934-19-X.

Stockinger K, Shalf J, Wu K and Bethel EW (2005) In: *Proceedings of IEEE Visualization 2005 Conference (VIS'05)*. pp. 167–174.

Subedi P, Davis P, Duan S, Klasky S, Kolla H and Parashar M (2018) Scalable data resilience for in-memory data staging. In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC18)*.

Tikhonova A, Correa CD and Ma KL (2010a) Explorable images for visualizing volume data. In: *Proceedings of PacificVis*. pp. 177–184.

Tikhonova A, Correa CD and Ma KL (2010b) Visualization by proxy: A novel framework for deferred interaction with volume data. *IEEE Transactions on Visualization & Computer Graphics* (6): 1551–1559.

Tu T, Yu H, Ramirez-Guzman L, Bielak J, Ghattas O, Ma KL and O'Hallaron DR (2006) From mesh generation to scientific visualization: An end-to-end approach to parallel supercomputing. In: *Proceedings of the ACM/IEEE Conference on Supercomputing (SC06)*. pp. 91–es.

Usher W, Rizzi S, Wald I, Amstutz J, Insley J, Vishwanath V, Ferrier N, Papka ME and Pascucci V (2018) libIS: A Lightweight Library for Flexible In Transit Visualization. In: *Proceedings of the Workshop on In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization (ISAV)*.

Vishwanath V, Hereld M, Morozov V and Papka ME (2011) Topology-aware Data Movement and Staging for I/O Acceleration on Blue Gene/P Supercomputing Systems. In: *Proceedings of International Conference for High Performance Computing, Networking, Storage and Analysis (SC11)*. pp. 19:1–19:11.

Whitlock B, Favre JM and Meredith JS (2011) Parallel in situ coupling of simulation with a fully featured visualization system. In: *Eurographics Symposium on Parallel Graphics and Visualization (EGPGV)*. pp. 101–109.

Ye C, Wang Y, Miller B and Ma KL (2018) XImage: Explorable Image for In Situ Volume Visualization. URL https://chrisyeshi.github.io/ximage-scalar/.

Ye Y, Miller R and Ma KL (2013) In situ pathtube visualization with explorable images. In: *Eurographics Symposium on Parallel Graphics and Visualization (EGPGV)*. pp. 9–16.

Ye YC, Neuroth T, Sauer F, Ma KL, Borghesi G, Konduri A, Kolla H and Chen J (2016) In situ generated probability distribution functions for interactive post hoc visualization and analysis. In: *IEEE Symposium on Large Data Analysis and Visualization (LDAV)*. pp. 65–74.

Ziegeler S, Atkins C, Bauer A and Pettey L (2015) In situ analysis as a parallel i/o problem. In: *Proceedings of the Workshop on In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization (ISAV)*. pp. 13–18.