1. *In Situ* Methods

The term "*in situ* visualization" has evolved into an umbrella term to cover a variety of methods for processing. Recently, a group of approximately fifty visualization scientists convened to formalize the terminology for describing different *in situ* methods [ins16], also known as the "*In Situ* Terminology Project." This group currently is characterizing *in situ* methods using six axes: integration type, proximity, access, synchronization, operation controls, and output type. This section describes these axes, drawing from ideas and discussion from the participants of the *In Situ* Terminology Project.

1.1. Integration Type

Several different methods are used to integrate visualization capabilities into running simulations. There are many examples of simulation developers creating and embedding their own visualization routines as part of the simulation system. Such implementations tend to be lightweight but unsuitable for reuse elsewhere. For more universal reuse of in situ visualization capabilities, there exist general-purpose libraries intended to be used by simulations to incorporate visualization routines. These libraries allow visualization capabilities designed by one group to be directly integrated into a simulation of another group.

There are also indirect methods to integrate in situ visualization with a simulation. One such approach is to use a shared protocol to indirectly connect the two components. Typically this happens through a middleware framework, such as ADIOS or GLEAN, where one or both of the components could be using the simulation data for purposes in addition to visualization. Another indirect integration method is function interposition where functions already used in the simulation are replaced by functions that do in situ visualization processing. For example, the simulation's function to write data to disk can be replaced, unbeknownst to the simulation code, with an alternate function that intercepts the data for visualization purposes.

1.2. Proximity

The proximity between visualization routines and the simulation code can greatly affect performance. Enumerating all possibilities for proximity is difficult, especially in the face of emerging architectures and deep memory hierarchies. The closest proximity for *in situ* routines is to share the same cores as the simulation, but even this basic configuration is complicated when considering how data is moved through the cache. The furthest proximity for *in situ* would be to send data to faraway nodes, possibly even to distinct machines (and possibly even to another continent). Points along this spectrum include architectural features such as burst buffers, local file systems, dedicated connections (e.g., PCI between CPU and GPU, NVLink between GPUs), etc. Further, it is important to note that visualization routines may run in multiple locations. A common example would be to run data triage routines on the same nodes as the simulation and also to run additional visualization routines on distinct nodes (that access data via a transport operation).

1.3. Access

An important description of an *in situ* system is its access to simulation data. With direct access, the visualization routine runs in the same logical memory space as the simulation code. In this case, the visualization routine typically gains access to data via pointers to simulation memory. With indirect access, the visualization routine runs in a distinct logical memory space from the simulation code. In this case, the visualization routine typically gains access to data via a communication mechanism that copies data from the logical memory space of the simulation.

Access is often conflated with proximity, because direct access occurs most often with on-node proximity, and indirect access occurs most often with off-node proximity. However, the remaining options are possible, although not common. Indirect access and on-node proximity occurs when visualization routines are run on the same nodes as the simulation, but using distinct memory resources (likely as a separate program running alongside the simulation). The remaining option, direct access and off-node proximity, can occur in PGAS-type settings.

1.4. Synchronization

Synchronization is about the relationship of "when" the visualization routines and the simulation code operate with respect to each other. With synchronous *in situ*, computing resources are devoted exclusively to the visualization routine or the simulation. In this model, the simulation and visualization routines trade off control of the computing resources, with only one executing at a time. With asynchronous *in situ*, visualization routine occurs concurrently to the simulation. In this model, the simulation and visualization routines can execute at the same time. This sharing may occur by partitioning compute nodes between simulation and visualization, by sharing resources within a node for both activities, or by other models where the allocation of resources vary over time.

As mentioned in the discussion of Proximity, visualization routines may be occurring in multiples locations within a single *in situ* system. In this case, each routine may have its own synchronization. Revisiting the example from the previous section of an architecture that does data triage in close proximity and visualization routines from distant proximity, it would be common for the former to run synchronously (i.e., the simulation passes execution control to the triage routine, which passes execution control back to the simulation when finished) and the latter to run asynchronously

(i.e., execute on data extracts after they arrive from the triage step).

1.5. Operation Controls

Operation Controls describe whether the end user can modify which visualization operations can be performed during execution. One type of operation controls allows the end user to modify the visualization operations being performed while the simulation is executing. This is often referred to as “interactive" usage. Interactive controls often have further distinctions regarding whether the simulation data can be modified (i.e., “steering") or not. Another type of operation controls requires that the set of visualization operations to be performed be fixed before the simulation begins, i.e., they cannot be changed by the user during execution. This is often referred to as “batch" usage.

1.6. Output Type

Output type describes what the *in situ* visualization routines generate. While the output of the execution does not affect the design of the system per se, many participants of the *In Situ* Terminology Project felt that it was an important descriptor of the system.

Explorable outputs are outputs that are useful for post hoc exploration, while non-explorable outputs are outputs from visualization routines that are not useful for post hoc exploration. These two options are best seen as extremes of a spectrum. If the output of the simulation is static images, for example renderings of isosurfaces, then that would typically be described as non-explorable. That said, animating these images over time may enable post hoc exploration, so even this simple example is fuzzy. Further, the Cinema system [AJO*14], which extracts images *in situ* for multiple visualizations, viewpoints, and time slices, and then enables post hoc exploration by providing an environment where users can explore data in a traditional manner (for example by animating images from different viewpoints to fly around a data set), is an example of an approach which produces images and yet is clearly explorable. Other important examples of explorable extracts are those that compress fields, for example using wavelets, and those that extract key portions or aspects of the data (for example subsetting or topology).