VIZSCHEMA – A STANDARD APPROACH FOR VISUALIZATION OF COMPUTATIONAL ACCELERATOR PHYSICS DATA*

S. Shasharina[#], J. Cary, M. Durant, S. Kruger, S. Veitzer, Tech-X Corporation, Boulder, CO, 80303, U.S.A.

Abstract

Even if common, self-described data formats are used, data organization (e.g. the structure and names of groups, datasets and attributes) differs between applications. This makes development of uniform visualization tools problematic and comparison of simulation results difficult. VizSchema is an effort to standardize metadata of HDF5 format so that the entities needed to visualize the data can be identified and interpreted by visualization tools. This approach allowed us to develop a standard powerful visualization tool, based on VisIt, for visualization of large data of various kinds (fields, particles, meshes) allowing 3D visualization of large-scale data from the COMPASS suite for SRF cavities and laser-plasma acceleration.

INTRODUCTION

Visualization is extremely valuable in providing better understanding of scientific data generated by simulations and guiding researchers in designing more meaningful experiments. Scientific models need to be compared with each other and validated against experiments. Consequently, most computational scientists rely on visualization tools. However, visualization and data comparison is often made difficult by the fact that various simulations use very different data formats and visualization tools.

Self-describing data formats are increasingly being used for storage of data generated by simulations. Such formats allow the code to store and access data within a file by name. The file storage system then takes care of developing an index for the data. In addition, the data can be decorated with attributes describing the units, dimensions, and other metadata for a particular variable. The self-describing formats now in use also help to deal with binary incompatibilities. Because different machine architectures use different binary representations for numbers, a binary file written by one processor may not be readable by another processor. Self-describing data file formats and interfaces ensure that the data is written in a universal binary format on all processors, and that software reading the data translates it to the appropriate architecture-specific format.

The Hierarchical Data Format (current version is HDF5) [1] and the NetCDF [2] format are in common use in the fusion, accelerator and climate modeling communities. HDF5 allows one to create a multi-tiered data structure inside of a file, so that one can create nested structures of groups and datasets.

Examples of HDF5 use include plasma physics codes such as VORPAL [3], a 3D plasma simulation code developed under development and Tech-X, and SYNERGIA [4], a multi-particle accelerator simulation tool developed at Fermilab. Both codes are actively used in the COMPASS SciDAC project [5]. Many other communities (earth sciences, fusion simulations) also use HDF5.

In spite of the fact that all these codes use self-describing data format, their files are organized very differently. They often do not share the node structure, do not agree on attributes, use different names for physically similar variables and store data in different structures. In other words, self-describing formats, though powerful, do not impose universally interpretable data structures.

For example, VORPAL put particles data in one dataset with all spatial information coming first: x = data[0,:], y = data[1,:], z = data[2.:], followed by momenta: p_x = data[3,:], p_y = data[4,:], p_z = data[5:,:], while SYNERGIA intermixes momenta and spatial information: p_x = data[0,:], x = data [1,:] etc.

How one can guess from looking at the data what is what? How does one recognize that a particular dataset represents a mesh and what kind of mesh is it? How does one indicate that a dataset is mapped to a particular mesh? Which data ordering is used (is it grouped by components or position indices)? Using some standards and common metadata within these formats could resolve this problem.

Visualization tools used by different teams are also very non-uniform. For a long time, scientific community used IDL [6] and AVS/Express [7]. Lately, many teams are moving towards the freely available, open source, high-quality visualization tools such as VisIt [8].

In this paper we present our efforts to develop such a standard for computational applications dealing with field and particles data. Our approach is based on first identifying the entities of interest to visualization, relationships between these entities and then defining intuitive and minimalistic ways to express them using metadata and common constructs used in self-described data formats: *groups*, *datasets*, and *attributes*. We call this data model VizSchema.

It is then used to implement a VisIt plugin (called Vs) which reads visualization entities from HDF5 files into memory and creates VisIt data structures thus providing a data importing mechanism from VizSchema compliant HDF5 files into VisIt.

In what follows we describe the VizSchema data model, Vs plugin, give examples of visualization and discuss future directions.

*Work supported by DOE grant DE-FC02-07ER54907.
[#]sveta@txcorp.com

VIZSCHEMA DATA MODEL

Principles

In this section we describe the elements of the VizSchema. These elements identify the data structures that one needs to expose in order to do visualization. They are not about HOW the visualization is performed (i.e. the type of light or position of the camera); instead, they are WHAT is being visualized (data and geometry) and WHAT needs to be exposed for minimal default visualization.

In designing the schema we use the following guiding principles:

- VizSchema assumes that data comes as one of three types: *variable*s (data which lives on a mesh described outside of the HDF5 node containing the data), *variables with meshes* (data which mixes physical values with the spatial information which is contained within the same HDF5 nodes) and *meshes.*
- These entities are identified by HDF5 markup and have particular attributes specific to their types.
- All the markup for the schema should be contained in the attributes so that users could choose the names of the data itself (typically contained in groups and datasets) as they please. The markup can be generated during I/O or added in a post-processing step. We expect these attributes to start with "vs".
- VizSchema attributes can refer to other entities using their short or fully-qualified names. If a short name is used, the reader will first search in the same space and then enlarge the search until the matching name is found.
- Each vs entity has an attribute vsType, which describes its category (variable or mesh, for example).
- Some entities have different kinds (i.e. subtypes), in which case a vsKind attribute specifies the kind.

Although, the schema entities described below use HDF5 lingo, mapping to the NetCDF lingo is straightforward; one needs just to substitute the term "variable" in place of "dataset." In the remainder of this section we give some details of the VizSchema elements.

Variables and Variables With Mesh

We assume that data comes as one of two kinds: a *variable* or a *variable with mesh*. A variable represents data, which lives on a mesh described outside of the variable array, while a variable with mesh contains spatial information within itself. In Particle-in-Cell simulations, all fields share the same mesh, so this mesh is described once and the values of the electric and magnetic fields do not contain the spatial information but rather depend on the tool to determine the mesh that they live on. Such fields are typically "variables." For particle data, one typically outputs their momentum and position in one dataset, so here the tool is supposed to generate a point mesh from within this dataset. So, particle data is a "variable with mesh." The suggested markup gives the information to the visualization tool to interpret the data. In the following pseudo-code snippet we show the variable markup in HDF5:

```
Dataset "phi" {
  Att vsType = "variable"
  Att vsMesh = "mycartgrid"
  Att vsCentering = "zonal"
}
```

The vsType attribute in this example indicates that this dataset needs to be visualized and needs a mesh called *mycartgrid* to be defined elsewhere in the file. The optional attribute vsCentering instructs that the data should be interpolated to a zone (with the default being nodal). The dimensions of the variable can be derived from querying the dataspace and are not needed in the explicit metadata.

Since variables with mesh mix spatial and other data in one dataset, there should be a way to specify the data structure. If the dataset's first N indices specify the coordinates (like in VORPAL), one could use the following markup:

```
Dataset "vorpalElectrons" {
  Att vsType = "variableWithMesh"
  Att vsNumSpatialDims = N
}
```

If the layout of data is different from this order (for example, like in SYNERGIA), one needs to use vsSpatialIndices, which would indicate which indices of the dataset contain spatial information:

```
Dataset “synergiaElectrons” {
  Att vsType = “varibaleWithMesh”
  Att vsSpatialIndices = [1, 3, 5]
}
```

Since the data can be ordered in many various ways, one also needs to describe the ordering of the data or the order of indices starting from the fastest-varying. For example, for the 3D case:

```
compMinorC = (i0, i1, i2, ic)
compMinorF = (ic, i2, i1, i0)
compMajorC = (ic, i0, i1, i2)
      (same as compMinorF for 1D)
compMajorF = (i2, i1, i0, ic)
      (same as compMinorC for 1D)
```

In component minor order, the indices (i0, i1, i2, ic) are such that the component index, ic, appears last. The C reference would be array[i0][i1][i2][ic], while the Fortran reference would be array(i0,i1,i2, ic). In component major, the indices (ic, i0, i1, i2) are such that the component index, ic, appear first. The C reference would be array[ic][i0][i1][i2], while the Fortran reference would be array(ic,i0,i1,i2).

When addressing the array in memory, two adjacent memory locations can differ by incrementing either the first index (Fortran) or the last index (C). Since the data is generally written to HDF5 files without changing the order, the component index must be specified. The default value of this attribute is compMinorC. This attribute is needed to reorder data as expected by a visualization tool.

Derived Variable

It is often useful to define additional variables, which are not being dumped by a simulation but present an interesting thing to see as well. That is why, in addition to the prime variable described above, we allow defining expressions using regular mathematical symbols. For example, one could define a density of electric energy as follows:

```
 Group anygroupname {
   Att vsType = "variableDefinition"
   Att vsDefinition = "elecEnergyDensity =
(E_0*E_0+E_1*E_1+E_2*E_2)"
 }
```

In defining this, we assume that the visualization tool can parse and evaluate such expressions. These assumptions are valid for our VisIt plugin implementation, which uses Python as its expression language.

Meshes

There is no uniform classification of meshes across tools and experiments. Based on our experience with several codes, we determined that the following mesh type categorizations are fairly general:

- Structured grid, which is defined by a list of points defined by their coordinates.
- Rectilinear grid, which is defined by the lists of increasing coordinate values for each axis and is a specialization of a structured grid
- Uniform grid (sometimes also called uniform Cartesian), which has constant distances between nodes in all directions and is a specialization of a rectilinear mesh
- Unstructured grid, which are defined by points and cells of various types.

The VizSchema markup for these mesh types is shown by the following examples. The first example describes a structured mesh with component-minor ordering. The dataset contains the mesh's points as an array ordered in X, Y, and Z, with 3 values (x,y,z) at each mesh point, for a total of 4 array dimensions:

```
 Dataset "mystructmesh" {
   Att vsType = "mesh"
   Att vsKind = "structured"
   Att vsIndexOrder = "compMinorC"
   Att vsStartCell = [0, 0, 0]
 }
```

The second example describes a 2D rectilinear mesh. It is a group containing 2 datasets, each of which contains the mesh points along one axis (X, Y). The optional vsAxis* attributes provide a name for each axis.

```
 Group "myrectgrid" {
   Att vsType = "mesh"
   Att vsKind = "rectilinear"
   Att vsAxis0 = "axis0"
   Att vsAxis1 = "axis1"
   Dataset axis0[n0]
   Dataset axis1[n1]
 }
```

The third example describes a 3D uniform mesh. Since all the mesh points are uniformly distributed, the coordinates of each point do not have to be provided. Instead, the VS attributes give the start and end position and number of cells along each axis, permitting a visualization tool to generate the mesh.

```
 Group "myunigrid" {
   Att vsType = "mesh"
   Att vsKind = "uniform"
   Att vsStartCell = [0, 0, 0]
   Att vsNumCells = [200, 200, 104]
   Att vsLowerBounds = [-2.5, -2.5, -1.3]
   Att vsUpperBounds = [2.5, 2.5, 1.3]
 }
```

The final example describes a 3D unstructured mesh. Such a mesh is generated from two arrays, one containing the coordinates of the mesh points, and the other containing entries giving the set of points that compose each cell in the mesh. By default, the coordinate array is named "points" and the cell array is named "polygons". The optional attributes vsPoints and vsPolygons permit arrays with non-default names to contain this information.

```
 Group "mypolymesh" {
   Att vsType = "mesh"
   Att vsKind = "unstructured"
   Att vsPoints = "points"
   Att vsPolygons = "polygons"
 }
```

The list of supported kinds of meshes will be growing as we encounter more kinds of simulation data. Some of them will need to have mappings to already existing types with the data translations implemented in the Vs plugin.

Multi-Domain Data

Quite often simulation data comes from multiple domains and uses different names in these domains, while it would be natural to treat it as one variable in a continuous domain. For such cases, we use vsMD attribute, which instructs visualization tools to connect data having the same value of this attribute.

Here is an example of two domain blocks that will be treated as a single multi-domain mesh named "edgeMesh" amd the two variables psiPriv and psiSol are declared to be an md variable named psi:

```
Dataset "privMesh" {
  Att vsType = "mesh"
  Att vsKind = "structured"
  Att vsMD = "edgeMesh"
}
Dataset "solMesh" {
  Att vsType = "mesh"
  Att vsKind = "structured"
  Att vsMD = "edgeMesh"
}
```

```
Dataset "psiPriv" {
  Att vsType = "variable"
  Att vsMesh = "privMesh"
  Att vsMD = “psi”
}
Dataset "psiSol" {
  Att vsType = "variable"
  Att vsMesh = "solMesh"
  Att vsMD = "psi"
}
```

Summary Of The Data model

To summarize, the visualization data model consists of variables, variables with mesh and meshes and their metadata. Variables metadata includes their names, their meshes, data ordering and centering. Variables with mesh have metadata for their name, data ordering, centering and separation of values from the spatial information. Meshes metadata depends on the mesh kind and fully describes each kind.

There are also variables defined as expressions and links that allow creating multi-domain variables.

In addition to the metadata described above, visualization needs additional metadata needed for correct allocation of the memory. For example, each dataset has its internal type (int, for example) and dimensions. This metadata should also be extracted before the visualization is possible but does not have to be present in the data markup.

VS PLUGIN

Based on the data model described above, we implemented a C++ data reader class, which reads all the needed metadata from HDF5 files into the memory. This reader creates an object that reflects the structure of an HDF5 file as it is seen by visualization – lists of variables with the meshes that they live on, variables with meshes, derived variables and meshes and all their metadata. Once such object is created, one uses the reader’s methods for reading these entities by their name. All the data is returned as a void* array (consistent with HDF5 model) for which memory should be allocated based on the metadata of this entity. The interface of the reader class is independent of the type of the visualization tool and is implemented for HDF5 data.

Next we created a VisIt plugin using the reader’s API. This plugin is available for the download at https://ice.txcorp.com/trac/vizschema/wiki/WikiStart. We are in the process of adding it to VisIt repository so it will be available upon VisIt installations.

EXAMPLES

Several codes adopted VizSchema and now provide the compliant output during I/O. One can also change the files after they have been generated using PyTables [9] (we have successfully using to change data as the schema evolved and also to annotate SYNERGIA files in accordance with the schema).

The plugin code was tested on Linux and OS X and is installed on such supercomputers as franklin.nersc.gov. Figs. 1-4 show some examples of visualizations done using the VizSchema plugin for VisIt. Fig. 1 is a screen capture of OASCR Award for Scientific Visualization at the 2008 Scientific Discovery through Advanced Computation Conference (Seattle) for the video, “Visual Inspection of a VORPAL Modeled Crab Cavity.”

Fig. 2 has been used as a cover for one of the issues of SciDAC review magazine [10]. Fig. 3 shows visualization for SYNERGIA data. Fig. 4 shows an example of multi-domain visualization and demonstrates that VizSchema is general enough to accommodate applications outside of computational accelerator physics: data from FACETS (Framework Application for Core-Edge Transport Simulations) [11].

CONCLUSIONS AND FUTURE DIRECTIONS

Standardization of the HDF5 output using consistent markup for visualization proved to be useful in accelerator physics applications as well as other domains having notions of fields and particles. The developed VisIt plugin is available for all interested parties.

In the nearest future we intend to extend the schema and the plugin with more detailed metadata for unstructured meshes and bring more applications into the VizSchema realm.

It will be interesting to develop a means to automatically annotate data with the markup. One could have a text or XML input for mapping internal data to the data elements of the schema and then use PyTables to add the expected attributes.

ACKNOWLEDGMENT

We would like to thank the VisIt, FACETS and VORPAL teams for helping us in this work. We also thank C. Geddes and G. Weber for providing us with images and J. Amundsen for providing us with SYNERGIA data.

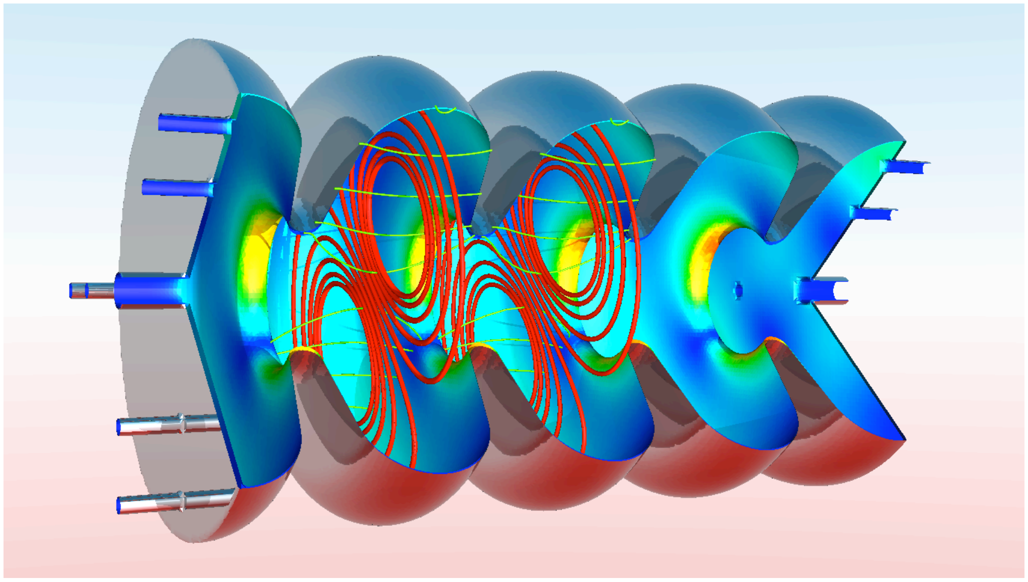

Figure 1: Examples of a visualization of VORPAL data: electromagnetic fields (red and green) and magnetic stress on the cavity (on the walls).

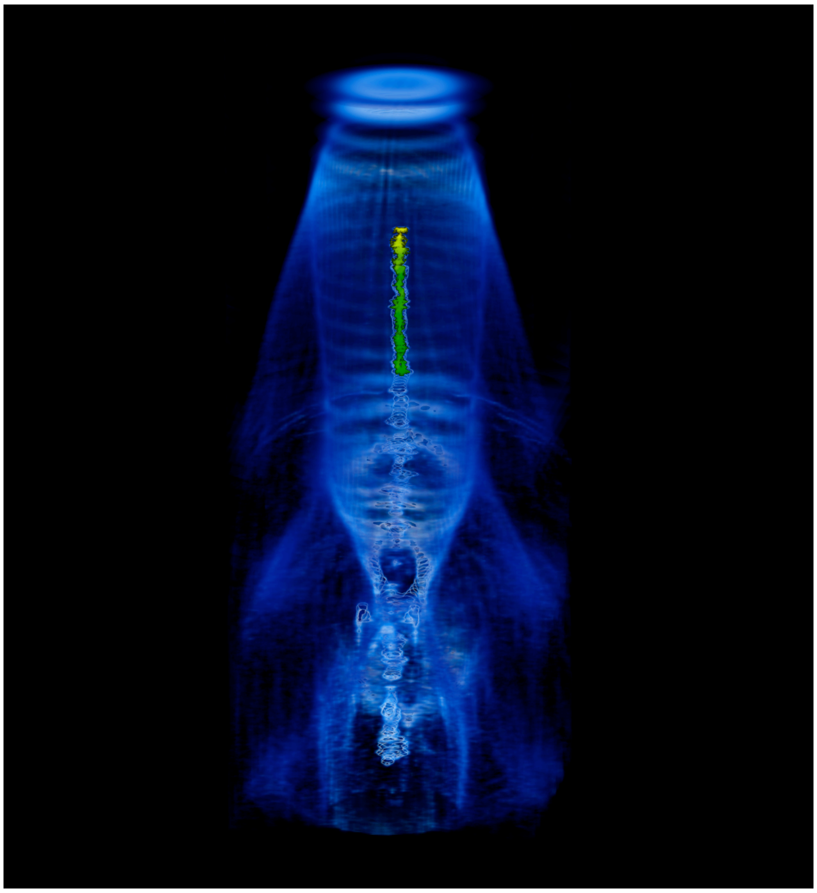

Figure 2: A three-dimensional VORPAL simulation models the self-consistent evolution of the wake resulting from a laser pulse and the acceleration of particles in a laser-plasma particle accelerator. Shown in volume rendering are the wake (blue) and a particle bunch (green and yellow). Courtesy of *G.H. Weber and C. Geddes.*

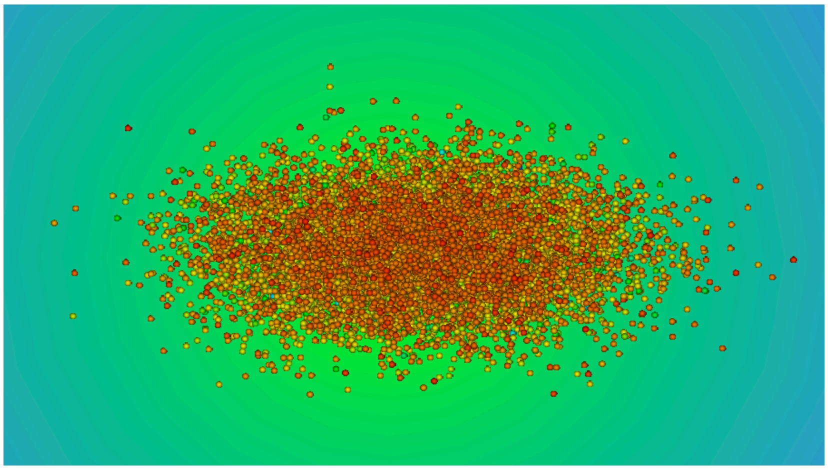

Figure 3: Examples of a visualization of SYNERGIA data: beam colored by the energy of the particles.

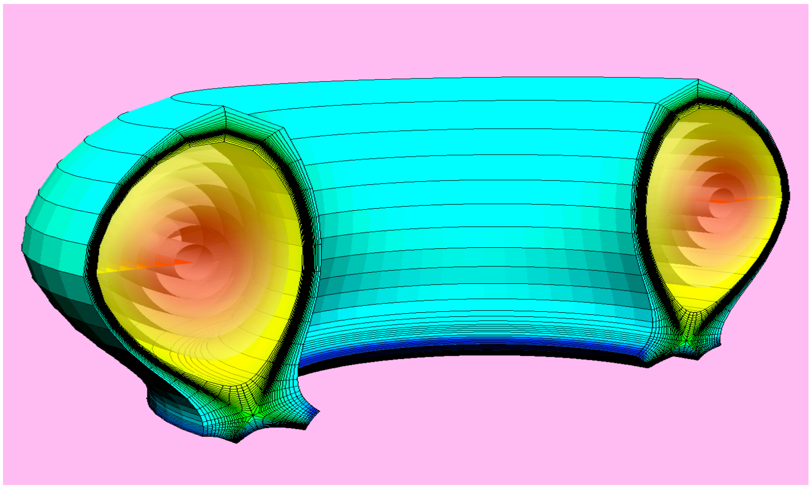

Figure 4: Examples of a visualization of FACETS data: electron temperature defined in multiple domains.

REFERENCES

[1] http://hdf.ncsa.uiuc.edu/HDF5/.
[2] http//www.unidata.ucar.edu/packages/netcdf
[3] C. Nieter and J/ R. Cary, "VORPAL: a versatile plasma simulation code," *J. Comp. Phys.* 196, 448-472 (2004).
[4] J. Amundson and P. Spentzouris, J. Qiang and R. Ryne, Synergia: A 3D Accelerator Modelling Tool with 3D Space Charge, *Journal of Computational Physics*, Volume 211, Issue 1, 1 January 2006, Pages 229-248.
[5] *http://www.scidac.gov/physics/COMPASS.html.*
[6] http://www.rsinc.com/idl/index.asp.
[7] http://www.avs.com/.
[8] H. Childs, E. S. Brugger, K. S. Bonnell, J. S. Meredith, M. Miller, B, J Whitlock and N. Max, A Contract-Based System for Large Data Visualization, *Proceedings of IEEE Visualization* 2005, pp 190-198, Minneapolis, Minnesota, October 23--25, 2005.
[9] http://www.pytables.org.
[10] http://www.scidacreview.org/0903/index.html.
[11] J. R. Cary, J. Candy, J. Cobb, R. H. Cohen, T. Epperly, D. J. Estep, S. Krasheninnikov, A. D. Malony, D. C. McCune, L. McInnes, A. Pankin, S. Balay, J. A. Carlsson, M. R. Fahey, R. J. Groebner, A. H. Hakim, S. E. Kruger, M. Miah, A. Pletzer, S. Shasharina, S. Vadlamani, D. Wade-Stein, T. D. Rognlien, A. Morris, S. Shende, G. W. Hammett, K. Indireshkumar, A. Yu. Pigarov and H. Zhang, Concurrent, parallel, multiphysics coupling in the FACETS project, *SciDAC 2009, J. Physics: Conf. Series* 180, 012056 (2009).