Integrating Visualization Applications, such as ParaView, into HEP Software Frameworks for In-situ Event Displays

A L Lyon, J B Kowalkowski, C D Jones
Fermi National Accelerator Laboratory, Batavia, IL 60510-0500 USA
E-mail: lyon@fnal.gov

Abstract. ParaView is a high performance visualization application not widely used in High Energy Physics (HEP). It is a long standing open source project led by Kitware and involves several Department of Energy (DOE) and Department of Defense (DOD) laboratories. Furthermore, it has been adopted by many DOE supercomputing centers and other sites. ParaView is unique in speed and efficiency by using state-of-the-art techniques developed by the academic visualization community that are often not found in applications written by the HEP community. In-situ visualization of events, where event details are visualized during processing/analysis, is a common task for experiment software frameworks. Kitware supplies Catalyst, a library that enables scientific software to serve visualization objects to client ParaView viewers yielding a real-time event display. Connecting ParaView to the Fermilab art framework will be described and the capabilities it brings discussed.

1. The Problem

3D visualization event displays in High Energy Physics fulfill many important use cases including,

- Analysis (e.g. event scanning).
- Interactive reconstruction.
- Algorithm development.
- Understanding and validating simulated events.
- Monitoring detector data.
- Public relations / Inspiration.

It is difficult for one application to satisfy all of these use cases. There are, however, common elements that all 3D visualization applications share,

**Display Event Data:** Hits, tracks, energy deposits, etc. with annotations (e.g. color or text labels indicating particle ID, energy, momentum, charge time, etc.).

**Display Detector In Background:** Necessary to locate the event data in 3D space. Detector may be a simple cartoon or a detailed picture.

**Interaction:** Zoom, rotate, pan, hide elements, select elements, view annotations, etc. The usefulness of the application often depends on the responsiveness of these interactions. Slow response yields a frustrating experience.
**BONUS** Data processing framework reaction: For example, select hits that are part of a track in the 3D event display, click a button, and redo tracking without those hits.

Recent years have seen a plethora of 3D visualization programs, especially with the introduction of WebGL[1] making writing in-browser 3D applications easier. For example, one particular experiment at Fermilab has seven visualization applications. A common difficulty that nearly all 3D visualization applications share is not natively understanding HEP data objects. This situation is to be expected: generic visualization libraries require data on shapes and polygons to draw. Unless HEP data comes with this information, and such data for Fermilab neutrino and muon experiments do not, then a translation step is required. Most of the web based applications require a JSON representation of the event and then do the translation to shapes at run time. Therefore an extra translation of HEP event data to JSON needs to occur with those JSON files needing storage, cataloging, bookkeeping, etc. Because the web based application is separate from a running HEP processing job, interaction back to the experiment framework is next to impossible.

Furthermore, in general many applications are written using basic WebGL or OpenGL calls without regard to graphics performance. For complicated events with many displayed objects, the user experience typically becomes frustrating due to lag between user input to manipulate the display (rotate, pan, zoom) and the resulting scene update. The only option to mitigate such lag is to display a subset of the data objects, and choosing those objects may introduce its own lag, compounding the level of frustration.

Finally, there are situations where a user may want to overlay visualization information from different sources. Examples include,

- Overlaying event information on top of simulation geometry
- Overlaying several events simultaneously
- Overlaying simulation geometry data with a 3D CAD model for comparison

The author is not aware of an HEP 3D visualization application that achieves the last example.

2. A Use Case
The Muon g-2 experiment at Fermilab[2] has a Geant4[3] based simulation of the storage ring and associated detectors, including 24 calorimeters spaced evenly on the inner side of the ring. The simulation geometry is very complex and high fidelity. Some validation tests of the calorimetry data in the simulation indicated that the calorimeters may be slightly misplaced. The calorimeter construction in Geant was very complicated with calorimeter placement depending on complex routines to determine the vacuum chamber shapes.

A CAD 3D model in Inventor[4] of the ring and calorimeters based on accurate drawings existed. Using the current Geant4 and other HEP based visualization tools, there was no easy way to compare the placement of the calorimeters in the simulation to positions in the 3D CAD model.

3. The ParaView Solution
ParaView[5] is an open source Department of Energy supported general 3D visualization application mainly developed by Kitware[6] with major contributions from many DOE and DOD laboratories. ParaView is based on the C++ Visualization Toolkit (VTK)[7]. ParaView can read in a huge number of file formats, including Inventor produced stereolithography .stl. Like all other visualization programs, it lacks the ability to directly understand the Geant4 geometry and event data. To remedy that problem, one of the authors of these proceedings (Lyon) wrote a ParaView plugin in Python[8] that converts HepRep[9] visualization output
to VTK polygon objects suitable for ParaView. See https://redmine.fnal.gov/redmine/projects/geanttovtk/wiki. HepRep is an XML format of shape and polygon descriptions that correspond to a Geant4 geometry and/or event scene and is fairly convenient to translate into VTK objects via Python. An important feature of ParaView is the ability to load multiple visualization sources and display them overlaid in the same coordinate system. The resulting image is shown in Fig. 1 where the calorimeter misplacement is clearly visible.

![Figure 1. A comparison of the Geant4 calorimeter geometry (red) and the 3D Inventor model (grey and blue). A misplacement of the calorimeter is clearly visible.](image)

ParaView’s ability to overlay visualizations from different sources is now an essential tool to easily discover these otherwise very hard-to-find problems. Furthermore, we now directly import much of our Geant geometry directly from 3D CAD models[10]. ParaView makes it very easy to ensure that the geometry is correct.

4. A Visualization Framework
For purposes here, let us concentrate on the visualization use cases (see sec. 1) of analysis, algorithm development and validation. There is no standard HEP 3D visualization solution, therefore we must make choices in technology and tools. As usual, we wish for a system that allows us to concentrate on physics, leaving the graphics and interactions up to the visualization application. We require a smooth user interface without frustrating lags even when displaying a complicated scene with many objects. Finally, the solution should not make it impossible to communicate between the visualization and data processing/analysis frameworks.

5. ParaView as a Visualization Framework
ParaView has some unique features,
• Strong support from the High Performance Computing community (most DOE HPC facilities have a visualization group that works with ParaView).

• Uses advanced visualization techniques to maintain a responsive interface and good user experience. For example a scene will be decimated (resolution reduced) during manipulation such as zoom, pan and rotation. Full resolution is restored once the interaction ends.

• Uses a pipeline for applying filters that can change the display. For example, one can generate surfaces, contours, and slices easily.

• Fully scriptable with Python. Custom filters and sources may be written in Python with Numpy. Furthermore, Matplotlib is included for 2D plots.

• Includes a client-server system called Catalyst[11]. With Catalyst, you can have a program that pushes VTK objects to ParaView clients on the network.

The Catalyst feature is especially interesting as it allows us to incorporate the HEP data object to VTK object translation in the data processing framework and push those VTK objects directly to a ParaView client, making an event display in real time with processing. For muon and neutrino experiments using the art[12] framework, we have developed such integration between art and ParaView using Catalyst in a package called artvtk.

Figure 2. See text for explanation.

Figure 2 shows the workings of using a possibly remote ParaView client obtaining data pushed from an art job via Catalyst. Physicists write dynamically loaded art modules that process HEP data for an event in the event store. Data in the event store is immutable. There are several types of modules shown in the diagram.
**Producer:** Extracts data from the event and can add new data to the event store. Producers run in user-defined order.

**Filter:** Same capabilities as a Producer, but can also return "true" or "false" based on some condition in the code. If a filter returns false, subsequent filters and producers are not run.

**Analyzer:** Extracts data from the event for processing, but otherwise the data and event store are immutable. Analyzers are used often to make plots and ntuples. They run after all of the Producers and Filters are finished with the event. All modules must be independent. The order of execution of Analyzers is decided by the framework.

**Service:** A globally accessible C++ object. Services generally manage resources, such as database connections, that are accessed by multiple modules.

The diagram in Fig. 2 shows tracks being created from hits and placed back into the Event Store. A filter called `LiveViz` is run to check if any ParaView client is actually connected to Catalyst. It is not worth wasting time translating HEP objects into VTK objects if no one is looking at the results. The `CatalystLive` service is an object that manages the Catalyst resource and has methods for asking Catalyst if a client is connected. The `HitVtkViz` and `TrackVtkViz` producers take the corresponding hit and track data out of the event and create the corresponding VTK objects. These VTK objects are pushed back into the event store but marked as "transients". Transient data are not serialized. `art` uses a Root[13] i/o system for serialization and this system cannot easily write the VTK objects. Instead, Analyzers are used to extract the VTK data from the event and, using the `Catalyst live service`, builds the payload of the Catalyst data to send to ParaView. Once processing for the event ends, the payload is sent to the ParaView client and the data are displayed. Alternatively, VTK data can be written to a ParaView native file and viewed later, not in real time.

6. Examples
Several examples are shown from the Muon g-2 experiment simulation. Figure 3 displays an anti-muon orbiting the storage ring and eventually decaying to a positron. Several photons are produced during this event. The particle tracks are converted to VTK line objects. VTK data objects can have nearly arbitrary metadata. The particle name is such metadata in this case. Notice how categorical colors are displayed to distinguish the particle type.

Figure 4 is similar to Fig. 3 but at a different angle and with the storage ring apparatus overlaid. The transparency for objects may be set within ParaView, as was done in this case to more easily see particles traversing the structures.

Figure 5 shows the user interface for ParaView. Furthermore, the particle momentum is shown as little arrows along the track. The Pipeline viewer shows the filters that were added to display the momentum. The multi-block inspector allows the user to turn on and off (or make transparent) different objects in the display.

7. Conclusions
3D Visualization of HEP event and geometry information is extremely important to fully understand, validate, and analyze HEP data. We have presented the rationale of a strategy centered around ParaView as a visualization application and VTK as a toolkit. We have integrated ParaView and in particular its client-server communications component, Catalyst, into the `art` framework to create a real-time 3D event display.

For the future, we are re-writing the ParaView Python plugin that converts Geant4 visualization data (in the form of a HepRep file) to VTK objects. We will instead write a C++ Geant4 visualization plugin that can push VTK objects directly to a ParaView client with Catalyst or directly write to a native ParaView file. Furthermore, we have started to try `artvtk` for LAr neutrino experiments. Finally, we plan to design an interaction protocol so a user can do
Figure 3. Muon g-2 display without geometry. Note the coloring by particle name.

selections or modifications to the data in ParaView and that information will be passed back to the data processing framework (e.g. art). The event will then be re-processed and re-displayed in accordance to the user manipulations from ParaView.
Figure 4. Muon g-2 display with geometry non-opaque
Figure 5. ParaView interface

References

[1] Khronos WebGL Working Group, www.khronos.org/webgl
[2] J. Grange, et. al., Muon g-2 Technical Design Report, FERMILAB-FN-0992-E, arXiv:1501.06858 [physics.ins-det]
[3] S. Agostinelli, et. al., Geant4 - a simulation toolkit, Nucl. Instr. and Meth. A 506 250 (2003)
[4] Autodesk Inventor, http://www.autodesk.com/products/inventor/overview
[5] U. Ayachit, The ParaView Guide: A Parallel Visualization Application, Kitware, 2015, ISBN 978-1930934306, www.paraview.org
[6] www.kitware.com
[7] W. Schroeder; K. Martin; B. Lorensen (2006), The Visualization Toolkit (4th ed.), Kitware, ISBN 978-1-930934-19-1, www.vtk.org
[8] Python Software Foundation, www.python.org
[9] J. Perl (2004), HepRep: a Generic Interface Definition for HEP Event Display Representables, http://www.slac.stanford.edu/~perl/heprep
[10] L. Welty-Rieger, Using 3D Engineering Models in a GEANT Simulation, These proceedings
[11] A.C. Bauer, B. Geveci, W. Shroeder, The Catalyst Users’s Guide v2.0, Kitware 2015, http://www.paraview.org/in-situ/
[12] C. Green, et. al., The Art Framework, J.Phys.Conf.Ser. 396 022020 (2012)
[13] R. Brun and F. Rademakers, Proceedings AIHENP’96 Workshop, Lausanne, Sep. 1996, Nucl. Inst. and Meth. A 389 81-86 (1997), root.cern.ch