AliEVE – ALICE Event Visualization Environment

M. Tadel, A. Mrak-Tadel, CERN, Switzerland

Abstract

ALICE Event Visualization Environment (AliEVE) is a general framework for visualization of detector geometry and event-related data being developed for the ALICE experiment. Its design is dominated by a large raw event size (80 MByte) and an even larger footprint of a full simulation–reconstruction pass (1.5 TByte). The main components of the framework on the application side are facilities for data and task-management. The presentation layer consists of GUI and visualization elements. Application core is minimal, providing access to registered data and hooks for execution of specific tasks. CINT scripts are used to steer data extraction and build-up of GUI elements. AliEVE introduces the concept of experiment software independent data representations attained via preprocessing of the data and canonization of its format.

INTRODUCTION

In high-energy physics experiments it is customary to use the term *event-display* to refer to a program used for visualization of detector geometry and event data. Such programs provide different levels of functionality and experiments usually develop several specialized applications.

The areas of event-display usage range from on-line monitoring to physics analysis and hypothesis testing. Trying to pull together all the requirements, one could say that event-display programs of a given experiment provide *visualization of* and *graphical user interface* (GUI) *to* detector geometry and event-related data including simulation records (kinematics, hits, digits), raw data, reconstructed objects (clusters, tracks, kinks, V0's, primary vertex) and physics objects (b-tags, Z0, H-candidates, etc). Additionally, interface to experiment's software framework, especially to reconstruction & analysis algorithms, is mandatory.

The event-display programs differ in their purpose as well and are used by people with very different backgrounds. Experts use them for visual debugging of mostly everything that was mentioned above, from electronics read-out and detector functioning to software algorithms. Visualization also aids the development of reconstruction and analysis algorithms as it is allows better understanding of actual problems and performances. Non-experts use event-displays to acquaint themselves with the detector, event structure and reconstruction algorithms. Finally, displays are used for presentations, demonstrations and for outreach activities where a complex problem needs to be introduced to a wider audience.

Apparently, there are many elements and many purposes to visualization programs in a HEP experiment. With the limited resources available for the development it is practically impossible to satisfy all the requirements with a single application. To avoid multiple implementations of the same functionality and to provide the elements needed by specific end-user applications we propose a common framework, called *ALICE Event Visualization Environment* (AliEVE).

AliROOT[1], the offline framework of the ALICE experiment, is firmly based on the ROOT data-analysis framework[2]. To benefit from the large code-base and user-community of the ROOT project it is natural to split the development of AliEVE into purely ROOT-based part and ALICE specific part.

ALICE is the dedicated heavy-ion experiment of the LHC. It will operate in the Pb–Pb@5.5 TeV/nucleon mode for one month per year. In this regime, the compressed raw-data size is 80 MB per event, about 50-times the event-size of other LHC experiments. A central Pb–Pb event has 60.000 primary tracks and the data produced by a full simulation/reconstruction pass amounts to 1.5 GByte. This includes, for example, 600 k simulated particles (1.5 M actually tracked), 150 M TPC hits, 3.2 M TPC clusters, 1.6 M TRD clusters and 16 k reconstructed tracks.

The data-size clearly poses technical problems related to reading speed and memory consumption. However, the really challenging part is to provide an extensible selection mechanism allowing visualization of relevant portion of the data. For example, drawing all the tracks and clusters in an event, while technically feasible with interactive refresh rates, results in images of little practical use but for color plates or outreach web-pages.

In the following the design of AliEVE is presented. As the implementation is still in the mid-development phase the article closes with the report on the current status of the project.

DESIGN OF ALIEVE

In this section we present the basic components of AliEVE and its application core. For each element we discuss what is provided by ROOT, what can be provided by a general framework and what needs to be provided by the ALICE specific codes.

Basic components

The goal of common components and paradigms is to provide ready to use solutions and to serve as a good code-base for concrete, specific implementations. ROOT already provides many components. Some of them can be used directly others can be implemented by simple extension of available functionality.

Data management. Event-displays load data from external sources and store it in memory in an internal representation, usually chosen so that it can be displayed efficiently. To allow interaction with the loaded data, modify its state or remove it from the application, it must be stored in a well defined manner. ROOT container classes and the `TFolder` class provide an adequate solution.

Based on its validity, one can divide the data into two categories: global data (detector geometry, coordinate grids, text markup) and event data. It is a frequent use case to iterate through a series of events or to have two events loaded simultaneously for comparison. It is beneficial to introduce concepts of *data-store* and *data-source*: a data-store contains the data that comes from a single data-source and allows its management in a coherent way. Directory structure with arbitrary depth is used.

The data-management is thus reduced to handling of a set of data-stores. Usually there is one global data-store holding the detector geometry and one data-store for each event that has been loaded. During iteration through an event sequence, one simply drops the data-store belonging to the old event, instantiates a new one and populates it with data. The global store containing the geometry is left intact by the operation.

Task management. During visualization, there are many common tasks that require non-negligible usage of system resources, including loading of data-sets, running selections on data and converting the data between different representations. The tasks can be limited by their input (e.g. retrieving data via the network), CPU (transforming a large data-set to a different coordinate system) or a combination of the two (running selection on a large ROOT tree).

To support execution of sequential and parallel tasks we introduce the concept of a *task queue*. Tasks in one task queue need to be carried out sequentially while several task queues can run in parallel. This introduces the need for thread-support and thread-synchronization devices like mutexes, read-write locks and condition variables. ROOT provides an OS independent thread API.

Relation of tasks to data-sources and data-storages must be well defined in order to make proper processing possible. In general, one associates a task with a given data-source and locks it for the duration of execution. During the finalization of the task, the data-source lock is released, the data-store is locked and the results are registered into the application.

GUI elements. ROOT provides a rather complete set of OS independent GUI classes derived from Win'98 toolkit. A set of medium-level widgets, like canvas, browser and GL viewer, is likewise available and can be sub-classed for more specific uses. It is of great importance that GUI classes are accessible via the CINT interpreter as this allows on the-fly modification of existing widget hierarchies.

Additionally, ROOT introduces the concept of *object editor*: for each class that requires user interaction one provides a corresponding editor class that exposes the object interface via a hand-written GUI. As application and visualization elements are objects anyway this provides a convenient service for fast construction of complex control systems.

Nevertheless there is still an obvious need for medium-level GUI elements that operate on a set of objects. They provide a gateway to a higher level functionality that can not be exposed via a simple object-oriented interface. For example, one can envision a track-selection GUI containing a set of cuts that retrieves matching tracks from a data-source and replaces the contents of a given data-store by the results. The framework can provide management of such GUIs and offer them to a user as a list of available options.

High-level composite UIs can be constructed to provide frequently used GUI layouts. We have implemented a top-level application window including a graphical view and data-store browser with an object-editor. Additional medium-level UIs can be spawned as floating windows. This serves both as a minimal, ready-to-use application for simple tasks and as a base for further extension. In the future we will try to modularize it and provide a more generic framework for high-level GUI composition.

3D graphics elements. ROOT supports 3D visualization in several modes, including rendering of objects via OpenGL [3]. There is excellent support for rendering of detector geometries in variety of styles and a rudimentary interface for display of poly-markers and poly-lines.

For legacy reasons, the communication between 3D viewer and the application is non-transparent and requires creation of intermediate objects. To overcome this limitation, we introduced a new mechanism for direct rendering via OpenGL. To endow a class with this capabilities the programmer must provide a GL-renderer class that inspects the original object and makes GL calls directly.

The prepared visualization atoms include sets of colored points, quads and boxes, a track, a list of tracks and a simple interface for displaying textures. Markup objects (arrows, rulers, labels) and a base-class for raw-data visualization for detectors with a regular segmentation (e.g. silicon detectors and calorimeters) are planned. From these elements composite objects for visualization of reconstructed physics objects can be build.

Picking is supported by ROOT on the object level but it will have to be extended for finer grained interaction with the object contents. For example, given a set of quads representing silicon-detector digits for a single module (one object), we would like to identify an individual digit when it is clicked upon. This will be implemented by adding a second-level picking procedure with special rendering function provided by the object itself for disambiguation.

Application core

The central entity of AliEVE is the application manager. It is simply a directory of components currently present with the functionality to instantiate and delete them and to expose them to other elements of the application and to the user. Data-stores, data-sources, task-queues and GUI fragments are therefore handled internally in a symmetric manner. But the interface to access them and start operations on them, differ from case to case.

CINT scripts are used for initial application bootstrapping. First a meta-level scripts are called to load the global data and to initialize the event source. These are registered into the application manager and therefrom available to specific scripts that are called afterward to load the required parts of an event. A general structure of such a script is:

1. obtain event handle from the manager
2. perform data extraction and instantiate visualization object(s), fill them and set-up their properties
3. register the visualization object to the manager

In a similar spirit the GUI fragments can be instantiated and registered. Thus all components of the application are instantiated from a set of loosely coupled scripts.

The scripts can be invoked in basically arbitrary combinations and top-level covering scripts can take care of more complex setups. Additionally, a GUI front-end for running scripts can be easily constructed and offer the user a set of major visualization options prepared by the experts.

Visualziation Summary Data

Before the decision to use CINT scripting extensively was made, we tried to obtain high-degree of code reuse by introducing the concept of *visualization summary data* (VSD). The idea was to use a small set of basic classes and repack ALICE event data into canonic trees for kinematics, hits, clusters, reconstructed tracks, V0s and kinks. In the process all data was converted to global coordinates, containers were flattened and summary for each track label was made. The good part was that we were able to reduce the data volume to 20% and use a simple set of tree queries to select the data from all the detectors. Tree selections are completely general as one can type any selection formula and also perform object post-processing by redirecting the selection into a list. Additionally, the visualization was completely independent of AliROOT. The bad part was the loss of references to original objects.

Even though the concept was dropped as the mainline solution it still has a strong appeal as one is often interested in visualizing a rather specific sub-set of event data. Furthermore, ability to visualize the data without the experiment framework has two important uses. First, it can be used on workstations/laptops where the experiment software is not or can not be (Windows, for most experiments) installed. Second, a specific VSD can be prepared for educational or outreach purposes and used by universities for lab-work by the students.

The support for creation and reading of VSDs is still present in AliEVE, together with a set of basic classes needed to represent standard objects in HEP. The VSD concept further benefits from usage of CINT scripts, especially since they can be packed into a ROOT file together with the data itself and shipped to other people for further usage or inspection of certain occurrence. We expect that VSD usage will become important as more users will start working on specific problems.

DEVELOPMENT STATUS

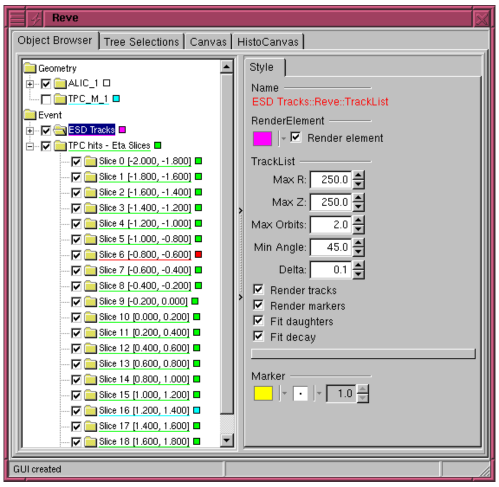
Prototype of AliEVE was constructed in the first half of 2005. It was implemented in the GLED framework [4] which provides object-collection management, multi-threaded method execution, auto-generated object-GUI and direct access to OpenGL (not present in ROOT at that time). With this functionality we were able to explore a wide range of algorithms in a rapid development cycle. VSDs were used for all but the raw-data visualization. ROOT's tree-queries proved to be an efficient and extensible selection mechanism. Many experiments were made with open GL and we have established that visualization of complete Pb-Pb events is feasible with standard graphics hardware if one uses all the available optimizations. The GLED prototype is still used for production high-resolution pictures and for outreach movies.

Minimal implementation within pure ROOT was finished in the beginning of 2006 and released to users by the end of April. The implementation of task queues is missing and GL interactivity is rudimentary.

Fig.1 shows a data-store browser filled with geometry and visualization objects. On the right side the object editor is presented, showing options for a track-list object (highlighted). Here one can set visualization parameters which apply to a whole track collection. Overloading of containers is used in many places to provide an interface to common properties of contained objects. GL rendering of the scene is shown in Fig.2.

Second GUI layout with interface to rendering of TPC raw-data is presented in Fig.3. The `TPCSegment` object contains the displayed data as well as the visualization parameters that can be changed dynamically in the object editor. The GL rendering class supports rendering via textures: in this case the whole TPC sector is rendered as three textured rectangles and thus provides a large speed improvement.

A major release with all the described functionality is



```
Geometry
  [✓] ALIC_1
  [ ] TPC_M_1
Event
  [✓] ESD Tracks
  [✓] TPC hits - Eta Slices
      [✓] Slice 0 [-2.000, -1.800]
      [✓] Slice 1 [-1.800, -1.600]
      [✓] Slice 2 [-1.600, -1.400]
      [✓] Slice 3 [-1.400, -1.200]
      [✓] Slice 4 [-1.200, -1.000]
      [✓] Slice 5 [-1.000, -0.800]
      [✓] Slice 6 [-0.800, -0.600]
      [✓] Slice 7 [-0.600, -0.400]
      [✓] Slice 8 [-0.400, -0.200]
      [✓] Slice 9 [-0.200, 0.000]
      [✓] Slice 10 [0.000, 0.200]
      [✓] Slice 11 [0.200, 0.400]
      [✓] Slice 12 [0.400, 0.600]
      [✓] Slice 13 [0.600, 0.800]
      [✓] Slice 14 [0.800, 1.000]
      [✓] Slice 15 [1.000, 1.200]
      [✓] Slice 16 [1.200, 1.400]
      [✓] Slice 17 [1.400, 1.600]
      [✓] Slice 18 [1.600, 1.800]

Name
ESD Tracks::Reve::TrackList
RenderElement
[✓] Render element
TrackList
   Max R:  250.0
   Max Z:  250.0
Max Orbits:  2.0
Min Angle:  45.0
   Delta:  0.1
[✓] Render tracks
[✓] Render markers
[✓] Fit daughters
[✓] Fit decay
Marker
   1.0

GUI created
```

Figure 1: Data-store browser and object editor.

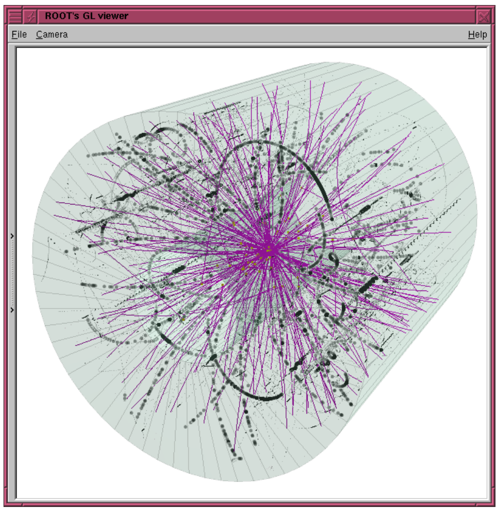


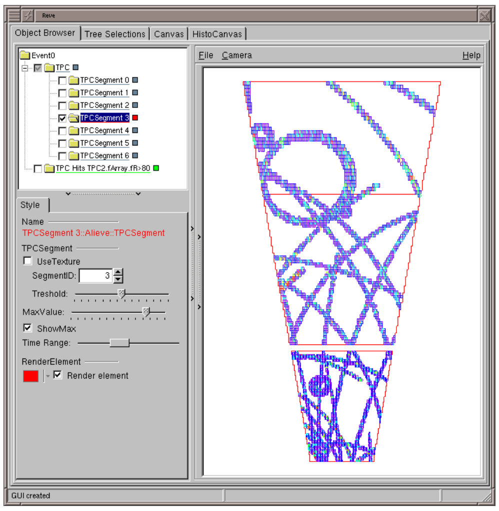
Figure 2: GL rendering of the scene from Fig.1

planned for September.

CONCLUSION

AliEVE framework provides the implementation of common visualization, GUI and application elements required by the ALICE experiment. Close integration with the ROOT framework and extensive reliance on CINT scripting minimizes application infrastructure and makes it easier to understand for developers and users alike. The GUI support follows the object-editor paradigm of ROOT for basic interface and build from it in a progressive, component-oriented manner toward medium and top-level elements.

Close integration with the ROOT framework and the concept of visualization summary data allows visualization to be decoupled from the experiment software framework and carried out on all platforms supported by ROOT. The preprocessing needed for data-conversion can also be used to reduce the data volume, if not all the data is required,



```
Event0
  [✓] TPC
      [ ] TPCSegment 0
      [ ] TPCSegment 1
      [ ] TPCSegment 2
      [✓] TPCSegment 3
      [ ] TPCSegment 4
      [ ] TPCSegment 5
      [ ] TPCSegment 6
  [ ] TPC Hits TPC2.fArray.fR>80

Name
TPCSegment 3::Alieve::TPCSegment
TPCSegment
[ ] UseTexture
  SegmentID:  3
  Treshold:
MaxValue:
[✓] ShowMax
Time Range:
RenderElement
[✓] Render element

GUI created
```

Figure 3: TPC raw-data visualization.

and to extend it with user-provided extensions.

ACKNOWLEDGEMENTS

Authors would like to thank René Brun and Federico Carminati for their guidance.

REFERENCES

[1] L. Betev, R. Brun, F. Carminati, P. Hristov, A. Morsch, F. , K. Safarik, *The ALICE Offline framework*, CHEP-2006, Mumbai, India.

[2] R. Brun and F. Rademakers, *ROOT – An Object Oriented Data Analysis Framework*, Proceedings AIHENP'96 Workshop, Lausanne, Sep. 1996, Nucl. Inst. & Meth. in Phys. Res. A 389 (1997) 81-86. See also `http://root.cern.ch/`.

[3] O. Couet, R. Maunder, T. Pocheptsov, R. Brun, *ROOT 3D graphics*, CHEP-2006, Mumbai, India.

[4] M. Tadel, GLED – *an Implementation of a Hierarchic Server–Client Model*, Applied parallel and distributed computing (ed: Y.Pan, L.T.Yang), Vol.16, Nova Science Publishers, 2005. See also `http://www.gled.org/`