HSF Visualization Workshop - 30 March 2017

Present:

Riccardo Maria Bianchi, Ilija Vukotic, Sergey Linev, Tai Sakuma, Tom McCauley, Matevz Tadel, Alja Mrak Tadel

---

Live notes:

Detector Geometry:

Tai: conversion to 3D binary formats and or text files formats like XML; possibility of using 3D packages like SketchUp (or others) to visualize the geometry. What about GDML? Is that standard?
Sergey: It is not used by all the HEP experiments but it is used outside of HEP as well, for example some CAD tools support that format. GDML is supported by ROOT as well.
Ilija: we want a format which supports tessellated geometry as well. Other requirements: if we want to get data from WEB, then we cannot use that kind of format: even if a text-based format like JSON is much more verbose, would be more convenient for Web applications. We could have multiple formats in use in parallel:

- JSON for web and by hand editing,
- binary formats:
    - tessellated objects - optimal for visualizations (eg. FBX)
    - storing primitives - for exchange with 3D modeling software eg. collada(.dae)

Sergey: JSON is editable by humans but you pay that in the size; JSROOT is transparent in terms of Javascript objects you get. TGeo classes are serialized in ROOT. Then you can transform TGeo classes in WebGL objects.
Ric: can we use ROOT to send data from one application to another one?
Sergey: JSROOT toward a web application, you don't need to install ROOT; to use it between two desktop apps you have to use the binary format, installing ROOT an all platforms, of course.

Sergey: TGeo was designed for simulation and geometry checks, but it's easy to add new shapes. For a new experiment is easier to use TGeo classes and add specific shapes they need, instead of building from scratch a new geometry library.

Ilija: ATLAS has a nice feature: shared instances used in generating volumes with functions
Sergey: TGeo has this feature as well: a shape is built once and then referenced when it is needed
Ilija: Do we want to support a bit deeper compression? In the sense: you give a function which generates positions and orientations
Sergey: There is this in TGeo
Ilija: do you store the function or the points?
Sergey: we store the function
Matevz: in TGeo we store shapes, logical volumes and mother-children relationships
Ilija: is there the possibility to define custom function to generate the geometry?
Sergey: it is not provided now, but it can be provided. So far, we provide a set of standard functions.

Event data:

Matevz: we can try to find common data among experiments; to try to build a common data format to be used in visualization
Sergey: it could be difficult with event data because they are live data; ROOT is equipped with JSON serialization
Matevz: we can provide common classes useful also for people and experiments not using ROOT; we can build converters from / to ROOT formats
Tom: for example, what if an experiment uses HDF5 files?
Sergey: we can have converters to Javascript and then do whatever you want

Alja: event data is very experiment specific. A calorimeter tower can be described as track or track as calorimeter. Cms also has dependency of what to visualize to the cms module descriptions
Matevz: we can simplify objects for visualization

TODO:

- Ask people from the WG to write a couple of paragraphs summarizing what they have now, what they think could go in a common data, and what they aspect for the future
- Or maybe summarizing into a paragraph per experiment
- We should try to attract other non-LHC experiments and communities

---

Chapter content:

Current Visualization landscape in HEP

- What we have now
- What issues we face
- Why we are proposing the following guidelines

Geometry description and visualization

Note: we should have a meeting with Simulation WG about this

- Geometry data: the scope
- Geometry formats
- Converters among formats
- Geometry hierarchy description
- Materials: density, … (needed for simulation as well)
- Visualization properties: colors, …
- Geometry delivery service API
    - API definition
    - Search/filter functionality
    - Level Of Details
    - Streaming

Event data description for visualization

- What is in the event: metadata, physics objects,
- We will try to define a common format for visualization purposes
- Strive for common format of standard HEP objects. Compose each class from mandatory, optional, and user-defined parts.
    - Simple tracks, …

  Eve contains a mini “Visuallization Summary Data” (VSD) set of classes that were used to make ALICE visualization decoupled from the AliROOT framework. Also, several EVE classes contain such information, e.g. TEveCalo, TEveDigitSet, TEveTrack classes, etc.
- A way to extend the format adding experiment-specific data/description for specific objects
- Goal is to have common tools to handle event for visualization
- Common tools provided by the community will handle the “common standard” part of the format, all other experiment-specific data will be handled by dedicated tools
- In this way will be able to use standard common tools for the large part of the tasks and they can maintain small in-house tools for specific needs

Truth data description for visualization

- Simulation purposes
- Description of truth data
- Converters from generators/simulation

Graphics libraries and technologies

1. Web-based
2. Application-based

1. 3D graphics frontends
    a. It seems there is interest in supporting the following:
        i. JSROOT
        ii. some other combination of three.js / web-gl / web-gui
        iii. Qt
        iv. Unreal and/or Unity

    Animation

User interfaces

GUI

Guidelines for GUI packages. Controllers (mouse & keyboard, touch screens, game controllers, positional controllers)

- Both desktop and browser-based UI libraries

CLI / API

- CLI: For example, we want to generated event displays from a list of parameters; or from a cron job (e.g. we want to generate event displays in control rooms automatically)
-

Generalized / “templated” GUI

There are several parts of GUI that allow inspection of or changing values of a given object. Those parts can be generated automatically, given instructions on allowed ranges / precision. The hard part are the callbacks … how user actions in GUI translate into actions. Obviously, each GUI frontend would require its own GUI generators from the common template.

Even though the top-level GUI is usually hand-crafted, prototype versions can be easily created using the templated GUI functionality.

We could provide a layer, a “glue” package, and plugin libraries behind for that; so we can replace the library behind if something changes in 5-10 years

Generation could be done at different levels, which will affect where you could use this meta-GUI info: web-based applications only or desktop applications as well.

GUI from data: “Next event” button shows up when many events are present

Multiuser

- Nowadays multi-users is used in many applications: for example GoogleDocs
- If we can provide a multi-users support for visualization, many users could explore and interact with events at the same time.
- Nice for expert users, important for Outreach & Education

Common HEP toolkits for Visualization

- “Glue” layer

Services & APIs for remote content delivery

Here we should separate interaction with a) a running experiment framework; b) some incarnation of a database / search engine.

- common parts of Geometry / Event data access
- Technologies: NoSql databases, search engines, cloud providers (eg. Google Big Tables)

Interacting with a backend

There are different use cases where it is necessary for a visualization application (browser based or standalone) to communicate with a backend. Examples: special event re-reconstruction, request to stage events, request to upload additional events into backend, etc. This necessarily requires very general API.

1. Data representation / formats

    a. For transport between server / client.
    b. This is now above in event-data description. As server side representation (like Eve classes, but can be more general, i.e. include more details).
    c. format for exposing the low-level experiment data format and available methods (what client can ask about). This is effectively a dictionary.
2. Data request / interaction protocol
    a. How to identify objects, containers
        i. Set of top-level, standard “entry” points, e.g. Configuration, Event, EventDB.
        ii. Sort of object hash / unique id is probably an easy optimization over URI
    b. Format of object detail request and response
    c. Format of method invocation request
3. Server / client state updates & synchronization
   Method invocation can have significant side effects. How do server and client figure out what needs to be transferred? One possibility:
    a. server notifies client with a list of changed objects
    b. client responds with a list of objects it is actually interested in
    c. server sends requested details

Support for Outreach and Education

- Requirements (multiplatform, open, easy deployment/update)
- Detailed visualization of particles propagating through a detector, showing all energy depositions, including calorimeter shower development.
- Multimedia content generation (video, photo, panoramic, audio,...)
- Collaboration with other groups (Machine Learning, …): providing simple event display tools to play/explore with simple data, for example for Machine Learning competitions (like the Kaggle one in the past) → Can be used for Open Data as well. In principle all LHC experiments are committed to release Open Data in the future, so if we have a common data format for visualization and a service to serve event data and geometry, we could in principle develop a community-developed display

Interaction with other projects

ROOT / ROOT-7, … others?
Simulation / geometry group
Data formats group
Reconstruction group