Event Visualization for EIC

Event Display v2 and beyond

Dmitry Arkhipkin
Brookhaven National Laboratory

STAR

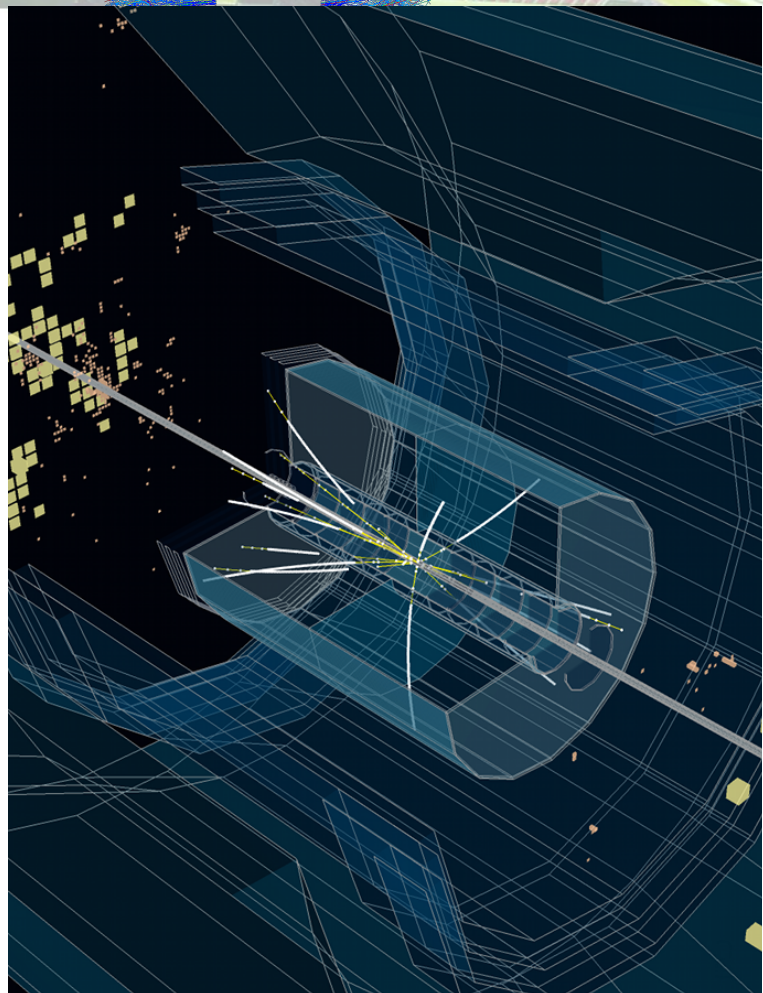
Office of Science
U.S. DEPARTMENT OF ENERGY

U.S. DEPARTMENT OF ENERGY

BROOKHAVEN NATIONAL LABORATORY

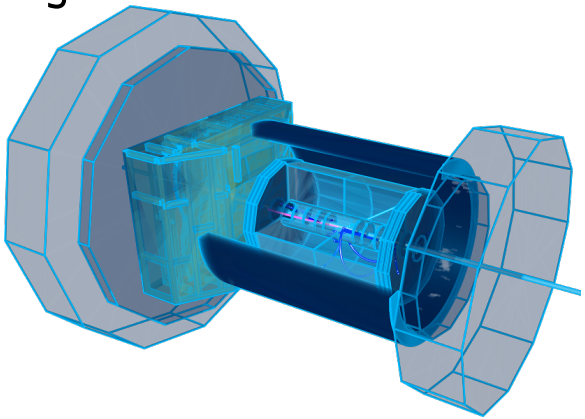
Outline

- Event Visualization Overview
- Use-Cases and Requirements
- Event Display v2
  - WebGL, Three.js and ES6
  - Internal Structure
  - Geometry Visualization Features
  - Event Data Visualization Features
- Future Plans
- Summary and Outlook



Overview

Geometry and Event Visualization



...for EIC detectors

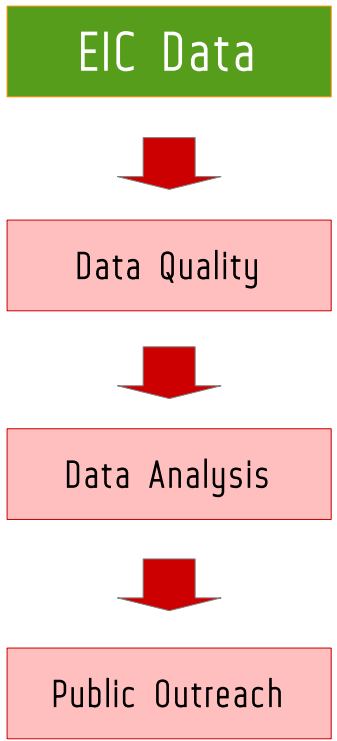
EIC Software Consortium focuses on the development of tools more advanced than those commonly available of the market today. Efficient event visualization ensures the cycle of detector R&D which goes faster and is better aligned with Consortium's goals. Assuming successful implementation, it will help to dramatically speed-up the visual design of the detector by EIC-involved scientists and engineers, modeling and analysis of detector data, will ease the visual interpretation and allow the cross-comparison of simulations results.

Generic Event Display will allow early optimization and debugging of the detector concept layout and geometry. It would also be a useful tool for developing the various reconstruction algorithms. Later on, when EIC detector will be commissioned, it will become a valuable tool for the real-time online monitoring of data quality, acceptance and event backgrounds, as well as a tool for detailed study of events of special interest. In addition to that, modern event display should be capable of providing high-quality images for public relations utility and for publications.

Use Cases

We have identified the following *user groups* and associated *use-cases*:

- *simulations and reconstruction experts* perform a detector geometry design and associated visualization, monte-carlo event data visualization, and require meta-data information access.
- *online subsystem experts* request fast reconstructed event visualization for early problem detection.
- *scientists performing data analysis* use reconstructed event visualization, detector geometry internals visualization, and require high-quality raster or scalable vector illustrations for papers and posters.
- *public outreach* needs are represented by STEM-teaching professors, media and news agencies, who would like to get a quick insight into the live experiment's life.



The extent of visualization use-cases implies that the visualization framework of an experiment must be able to handle all kinds of experiment data and present them together in a unified environment. Visualized event data should include tracker hits (3d points, TPC-like), calorimetric hits (blocks, or trapezoids for projective geometries), tracker pixels/pads (2d plane-like objects), and reconstructed physics observables like particle tracks, track vertices, V0s, electromagnetic showers and accompanying meta-data information – particle momentum, charge, energy deposition, track origin coordinates and much more.

Formal Requirements

Interoperability: minimize input geometry formats – single source of detector geometry information must be sufficient to perform simulation, reconstruction, online processing, data analysis and event visualization. Also, minimize input event data and meta-data formats.

Platform-independence: event display application should be supported on all commonly used platforms and operational systems. Currently, web-based client code and decoupled (via MQ) server code are under review.

Ease of Use: unified event display should have a minimalistic, simple and intuitive graphical user interface; associated learning curve should be negligible.

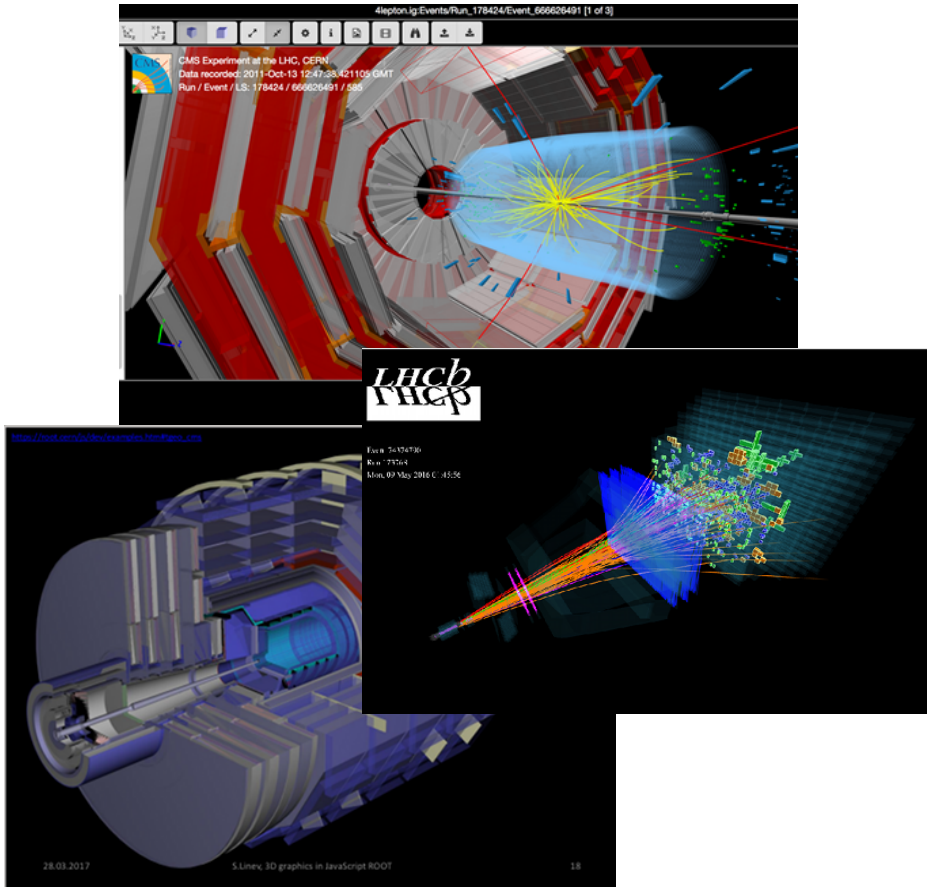
Interactive Features: it is expected that event display client package will provide an extensive interaction possibilities, covering vast array of user needs.

Loose Coupling: event display components should not carry a dependency on specific simulations or data analysis frameworks, and should be modular to allow easy integration in future.

Extensibility: it is expected that event display will allow optional customization of the generic display to the needs of the specific experiment via modules and a certain set of exposed interfaces, thus guaranteeing the minimization of future development effort.

...and more!

Vizualization Market Study



HSF Visualization workshop, March 2017
https://indico.cern.ch/event/617054/

Many experiment-specific Event Displays were presented, most started development around ~2014. Predominantly, they use a web-based event display implemented with the help of WebGL + Three.js + custom data format for geometry + custom format for events – exact same approach as STAR used since 2011.. Qt 3D is also there, but in very limited quantities.

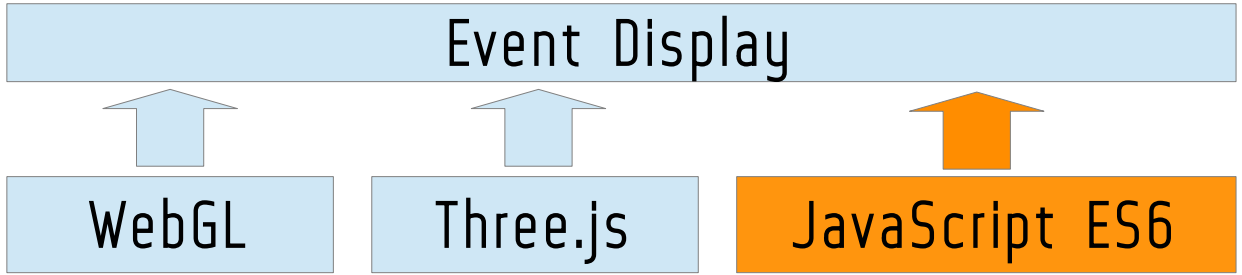
EIC visualization goal: generic format for geometry + generic format for event data -> STAR developed Event Display v2, which generalizes most components and introduces experiment-agnostic core tools.

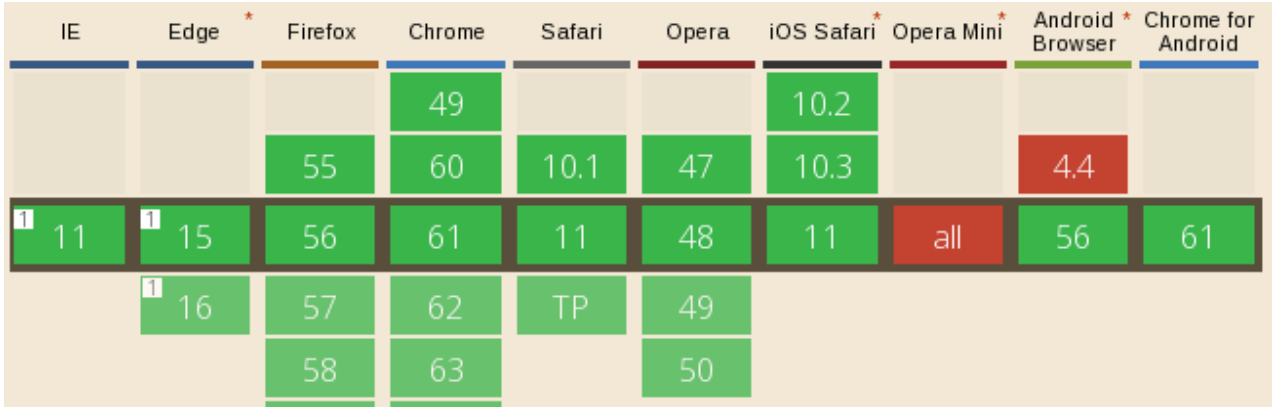
WebGL, Three.js and ES6

Technology of choice: WebGL, Three.js, JavaScript ES6

- WebGL: aka Web Graphics Library, is a JavaScript API for rendering interactive 3D and 2D graphics within any compatible web browser without the use of plug-ins.
- Three.js: is a high-level cross-browser JavaScript library/API used to create and display animated 3D computer graphics in a web browser. Three.js uses WebGL.
- JavaScript ES6: ECMAScript 6 (ES6) is the upcoming sixth major release of the ECMAScript language specification. ECMAScript is the "proper" name for the language commonly referred to as JavaScript. Could be transpiled to ES5 via Babel transpiler.

Cross-platform solution: web and Win/Lin/Mac/iOS/Android binaries from a single source - hybrid apps, Cordova





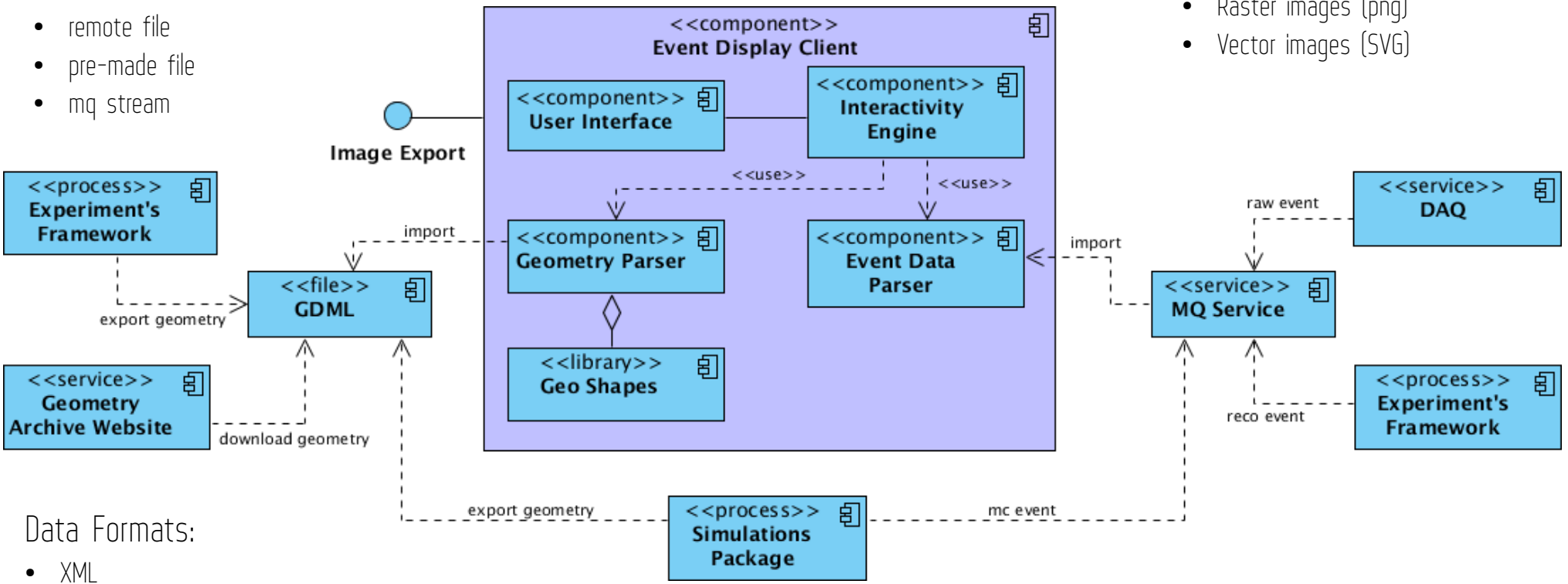
Can I Use: WebGL

Event Display v2: internal Structure

Import:

- local file
- remote file
- pre-made file
- mq stream

Component Scheme of the Event Display v2



Export:

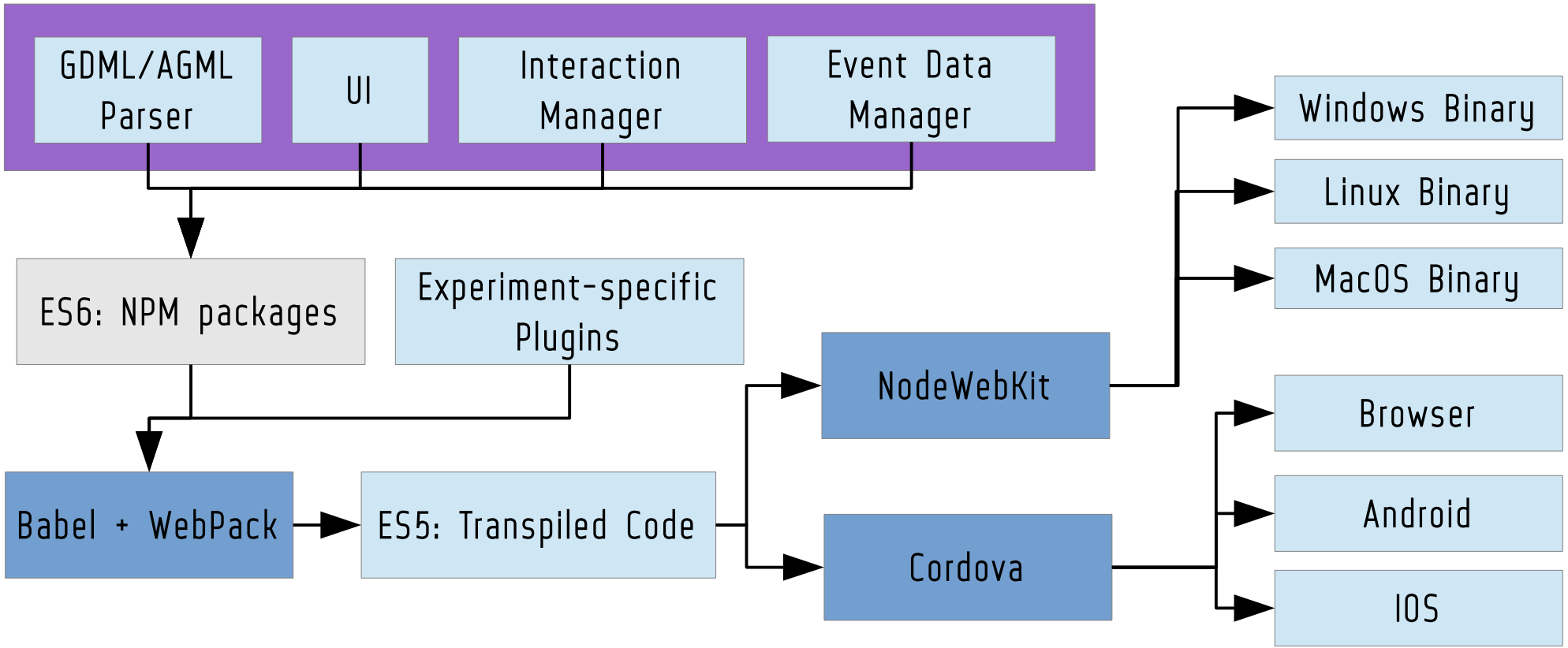
- Raster images (png)
- Vector images (SVG)

Data Formats:

- XML
- JSON

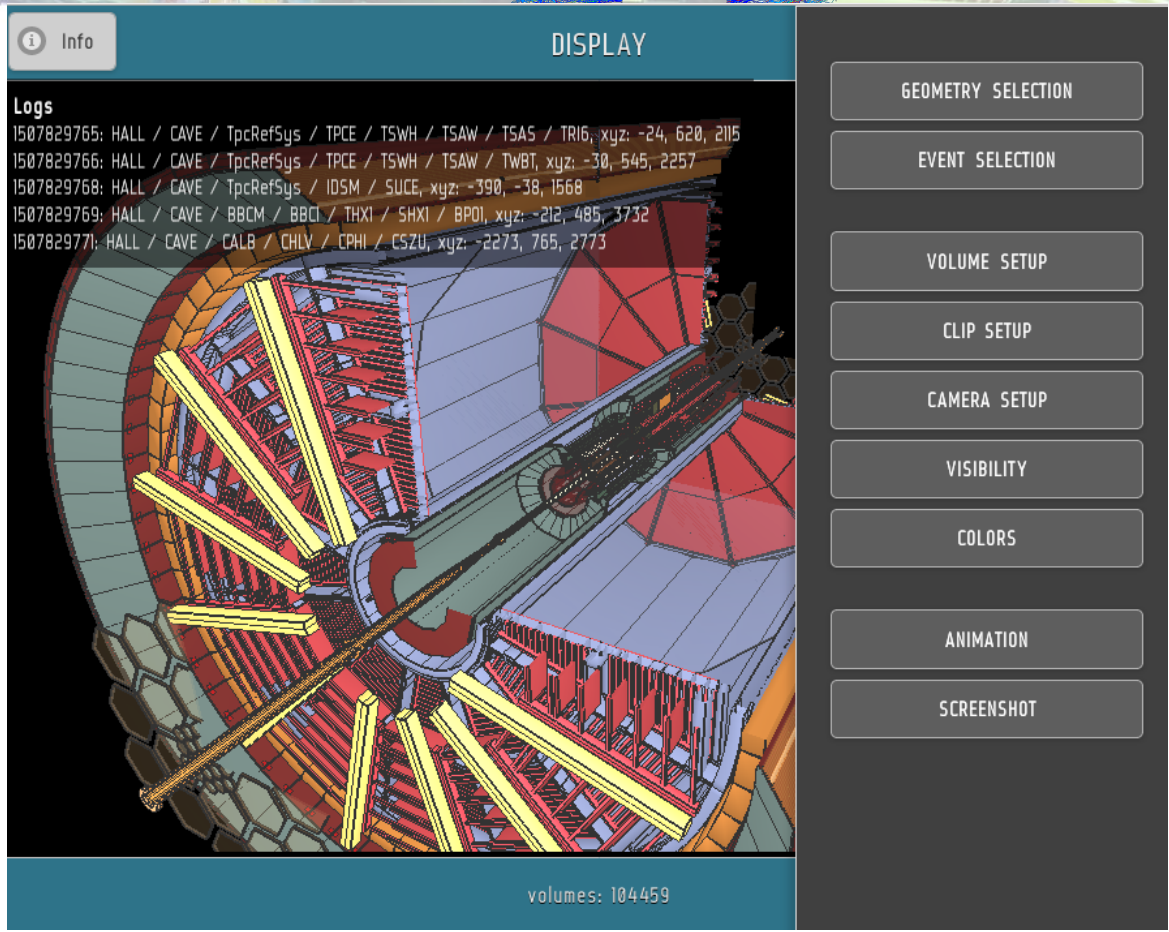
Event Display v2: internal Structure

Build Chain of the Event Display v2

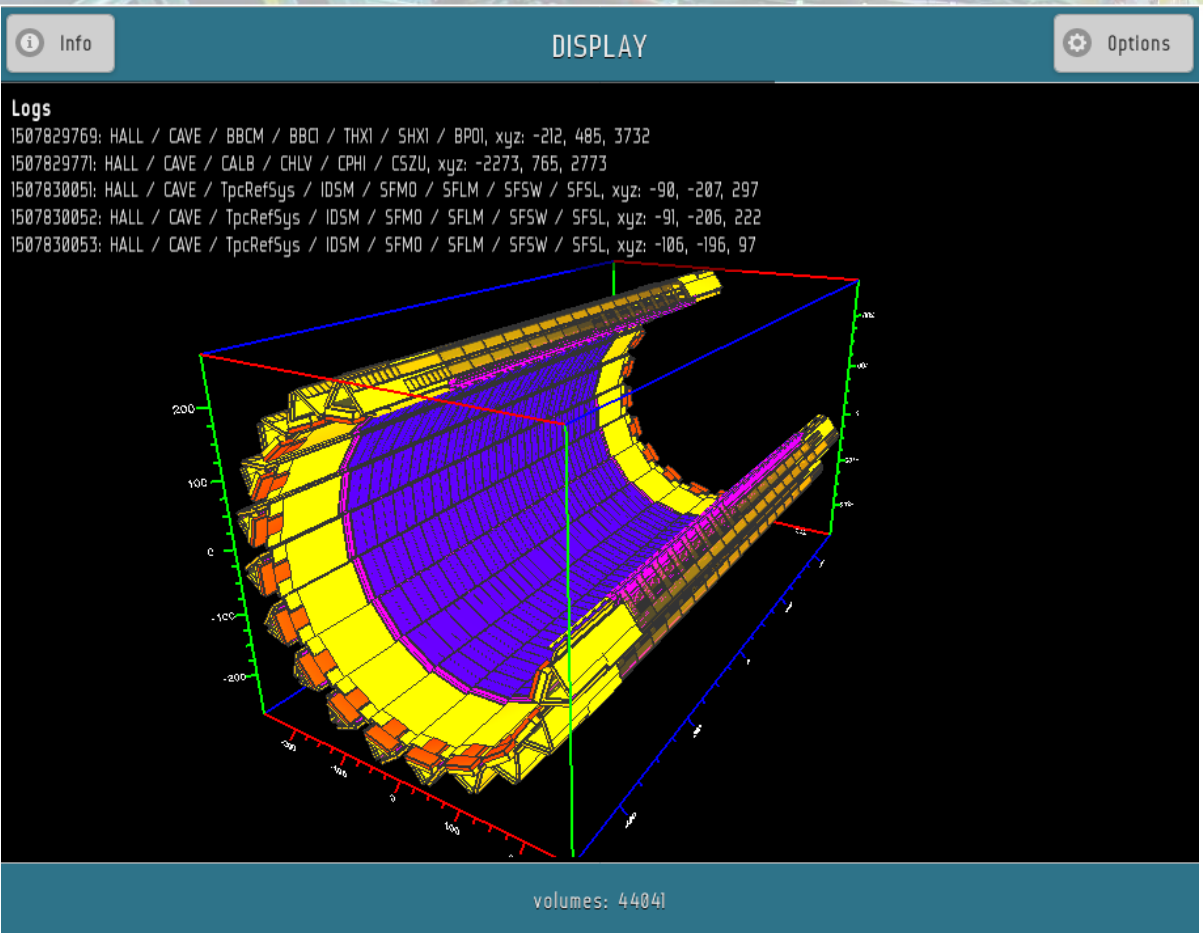


Geometry Visualization Features, I

- Input format:
  - latest GDML version, ~97% implemented, missing matrix operations and loops over volumes, multiple file support..
- Geometry Shapes:
  - 100% coverage of GDML/G4, TGeo, VecGeom
- Interactivity:
  - subselection of volumes
  - automatic volume positioning - parent matrix taken into account even if parent is not visualized
  - ... and more on next slide



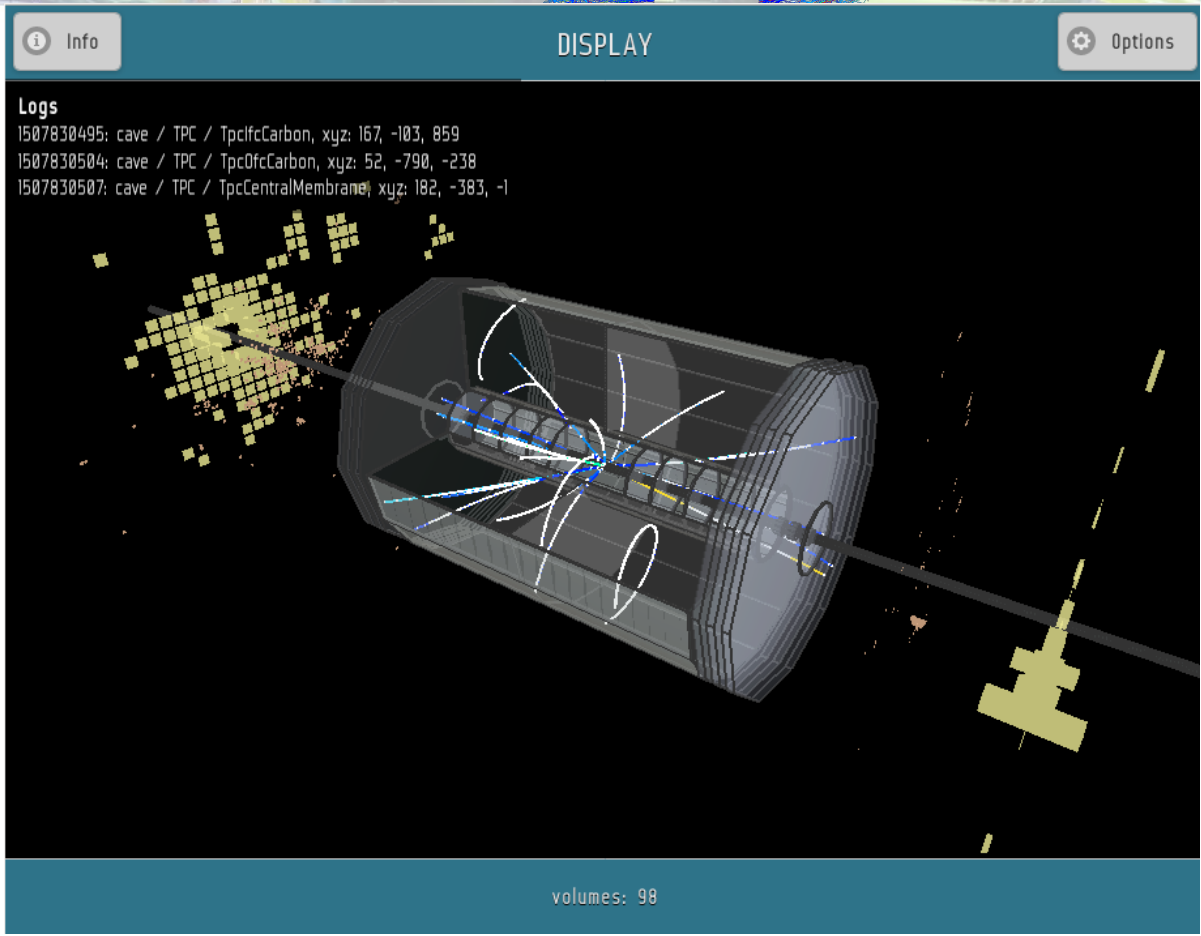
Geometry Visualization Features, I



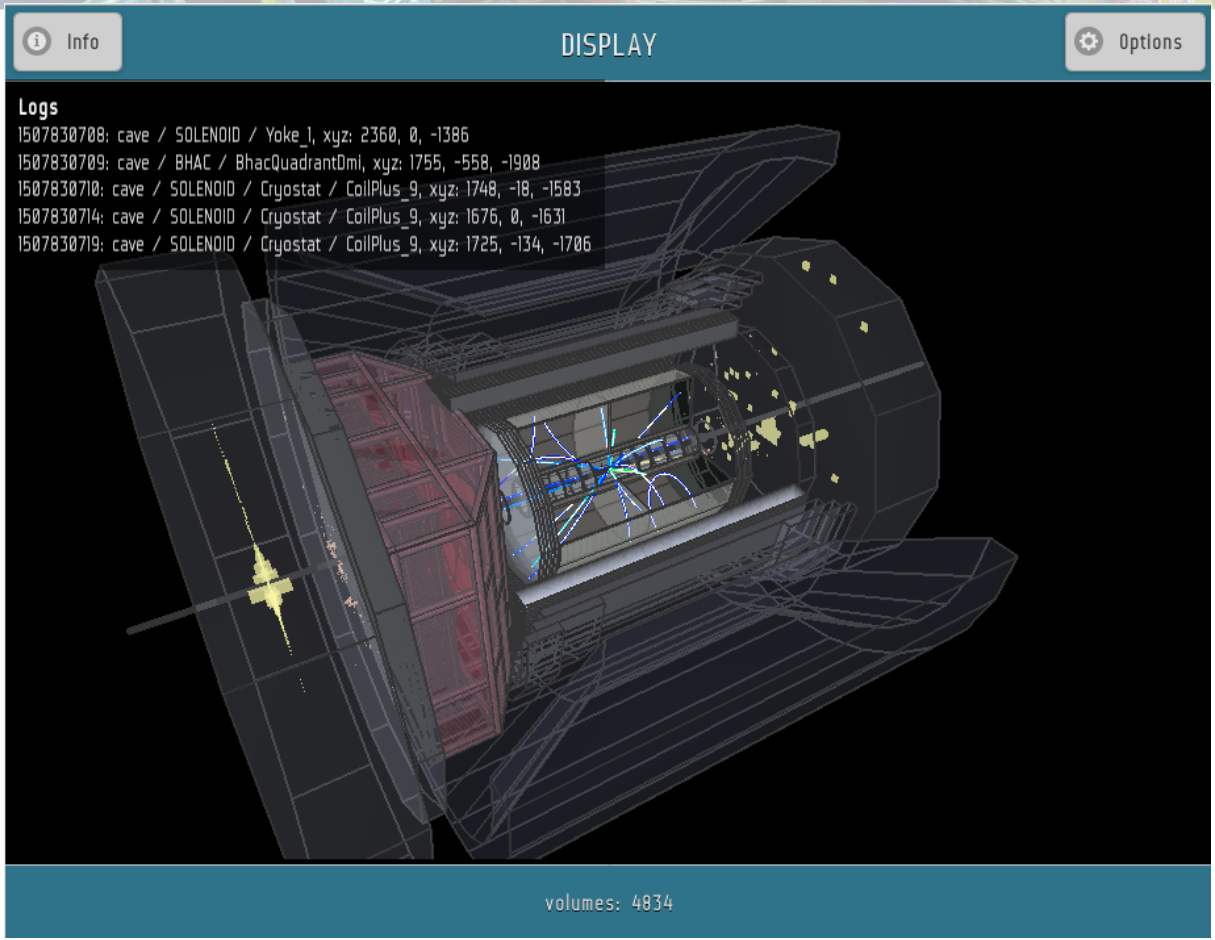
- More interactivity:
  - volume clipping
  - camera positioning
  - bounding box / arrows
  - configurable color palette(s)
  - automatic rotations
  - mouse-over volume ID display volume position information
- Export formats:
  - raster PNG, 8k by 8k pixels
  - vector SVG - high-quality publication images!

Event Visualization Features, I

Track objects: Event Display v2 supports two track serialized representations. First representation expects event tracks to be exported as a set of 3d points, connected by straight line by the display application. This ensures the independence of the magnetic field configuration. Second representation assumes uniform solenoidal magnetic field (field map support to come later) and negligible track momentum loss over the lifetime of the track. It expects tracks to be characterized by starting point of the track (x, y, z), initial momentum (px, py, pz), charge (+1,-1) and total track length. In this representation, we create a physical helix out of the input parameters, and project track away from the starting poing using Runge-Kutta propagator.



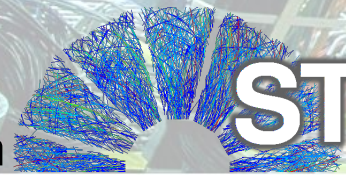
Event Visualization Features, II



Hit objects: Event Display v2 supports three hit representations: (a) TPC-like 3d points, (b) projective calorimetric hits and (c) non-projective calorimetric hits. First type of hits ("a") is defined by just [x,y,z] coordinates of the hit, and does not specify a volume shape. It is implemented as a set of eight-vertex boxes. Projective calorimetric hits ("b") require the following set of parameters:
[eta,phi,e,angle,volume name]. Visually, they are displayed as a set of trapezoids, covering specific solid angles as defined by eta,phi of the hit. Non-projective calorimetric hits require the following information: [x,y,z,volume name]. Event Display requests volume shape from the geometry module, and positions hits according to the provided parameters.

Event Visualization Features, III

Examples: Hit Geometry Definition, Track Data, Calorimetric Hit Data



STAR

JSON

```
<....>
        "TPC": {
            "type": "3D",
            "options": {
                "size": 5,
                "color": 16777215
            }
        },
        "BEMC": {
            "type": "BOX",
            "options": {
                "dx": 10,
                "dy": 10,
                "dz": 40,
                "axis": "z",
                "scaleminmax": true,
                "scalecut": 0.0,
                "color": 16763551,
                "transparent": 0.5
            }
        },
<....>
```

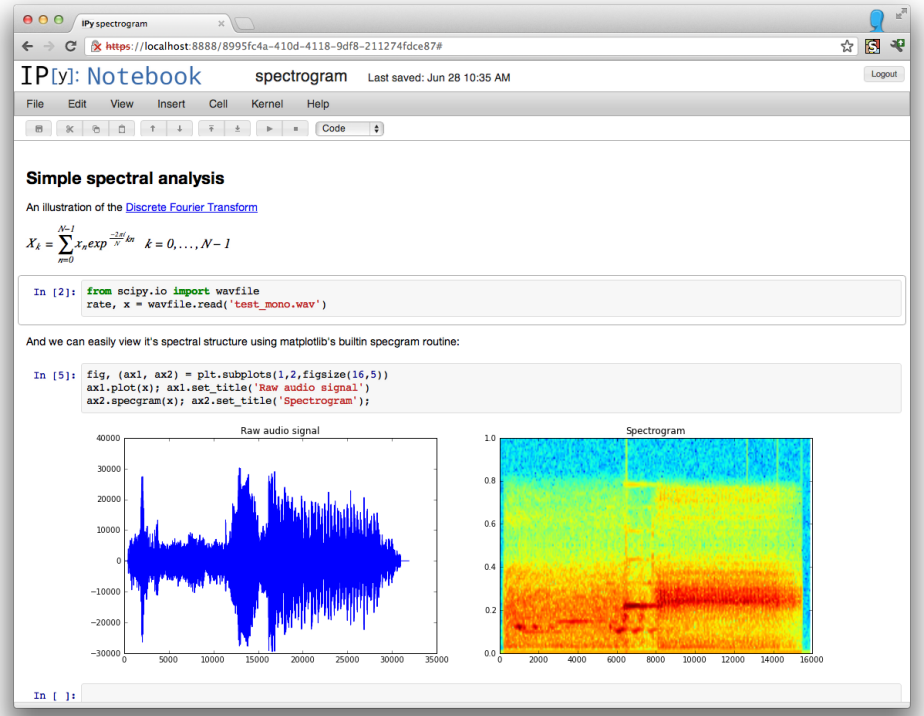
```
<....>
"TPC": [
  { "pt":     3.530, "color": 16711680, "t": 1, "e": -1.389, "p": -2.282, "c": -1, "q": -1, "nh": 70,
    "xyz": [    0.000,    0.000,    0.000],
    "pxyz": [   -2.305,   -2.673,   -6.636], "l":  131.240,
     "pts": [
       [ -1.488, -1.728, -4.287], [   -3.102, -3.611, -8.950], [ -8.633, -10.124, -25.014],
       [ -14.533, -17.177, -42.301] ,[ -14.832, -17.539, -43.183 ]
     ]
},
"BEMC": [
  { "x":    23.346, "y":   -20.889, "z": -165.000, "e":     0.004 },
  { "x":    23.346, "y":   -23.346, "z": -165.000, "e":     0.094 },
  { "x":    23.346, "y":   -25.804, "z": -165.000, "e":     0.012 },
<.....>
"EEMC": [
  {"e": "0.01", "eta": "0.02675", "phi": "1.28142" },
  { "e": "0.03", "eta": "0.075", "phi": "1.28142" },
  { "e": "0.11", "eta": "0.125", "phi": "1.28142" },
<.....>
```

Future Plans, I

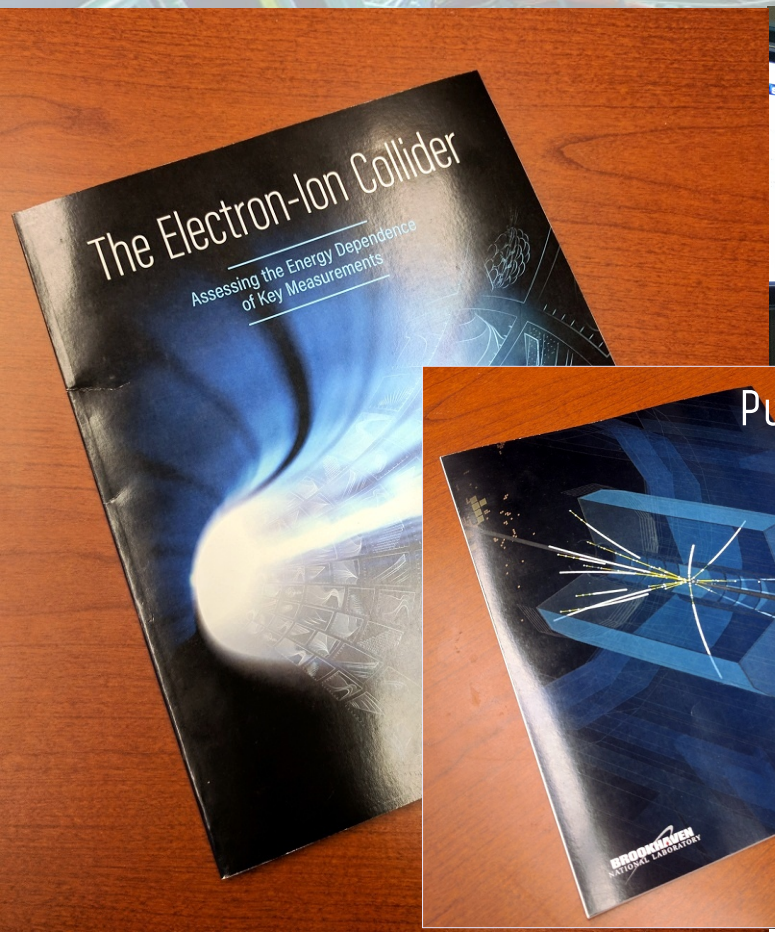
- Geometry Data
  - GDML/AGML support is great, but what about other formats? Anyone with LHC experience?
  - Chicken and the Egg problem: GDML first (GDML imported), or Framework first (GDML exported)? If former is the future, is there a need for a better geometry authoring tools?
  - Library of Geometries: single merged/independent files or large numbers of small files with dependencies? How to do dependency tracking in GDML-based geometry versioning? Any suitable storage for that apart from GitHub/GitLab?
  - Reusable geometries? i.e. combine TPC of BNL with Inner Tracker of Jlab for simulations purposes?
- Event Data
  - Need to converge on some generic event format, covering most event entities
  - Should be descriptive yet compact enough to allow streaming (mq) from server to hundreds of clients simultaneously
  - Need to develop a "standard" event manipulation API to allow certain level of interactivity (apply cuts, sort, use specific view etc)
  - Need to provide the example code of the exporter to that format (Framework-specific? Generic ROOT? Generic C++?)
  - Multi-user cooperative event exploration? (potential use in Universities)

Future Plans, II

- Several ideas are in the air right now:
  - Web-based GDML/AGML editor or even IDE – convenient way for users to author GDML/AGML files via accessible web interface. Schema validator, Wizard-like helpers, compatibility warnings (TGeo vs G4) and much more..
  - Jupyter Notebook integration – visualize detectors using original GDML/AGML files to illustrate your data analysis or special cases



Event Display Usage Examples



Summary & Outlook

- Event Display v2 Update:
  - Progress update on the Event Display v2 has been presented;
  - Tons of code updates implemented since May: better GDML support, core manipulation; functionality implemented, better 3d performance, new export formats. Still, many improvements to come by Christmas (or Spring, depending on how RHIC Run 18 cool down goes);
  - Getting closer to the official release of reusable "building blocks" for a generic Event Display!
- Near-Future Plans:
  - 100% GDML support by ~January 2018, start AGML import module;
  - Focus on the event description and event meta-data format research and development – need better generalization, more use-case coverage and feedback from users;
  - finalize ED as a set of reusable modules and upload to GitHub and NPM so others can try it;
  - Prototype GDML IDE and investigate Jupyter Notebook integration? No ETA here..

Thank You!

Questions? Comments?