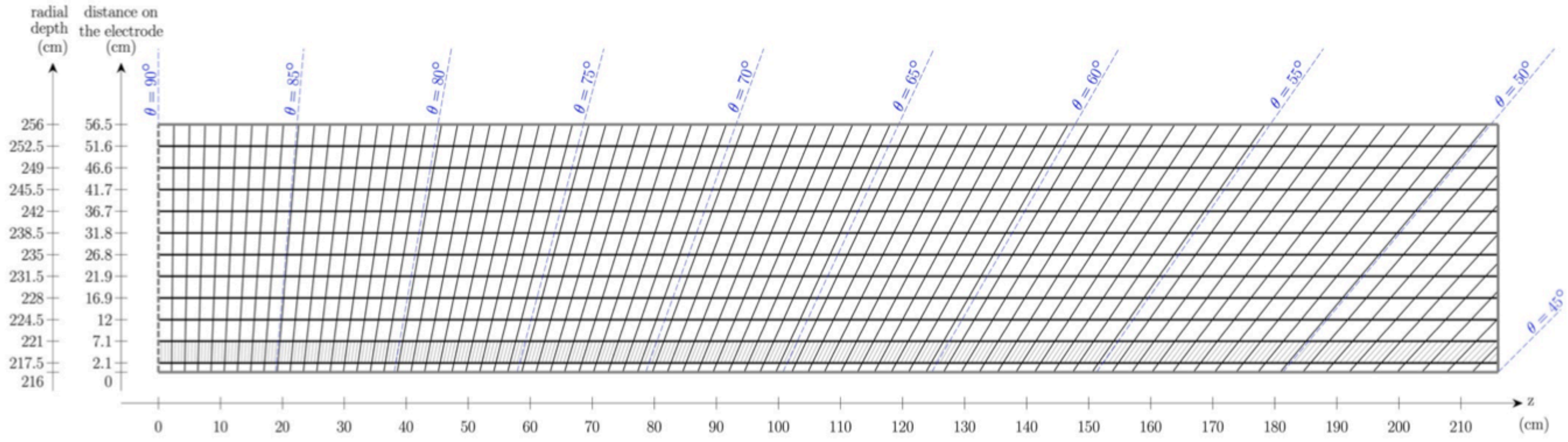
A ROOT-based event display to debug the FCC-ee LAr calo reconstruction

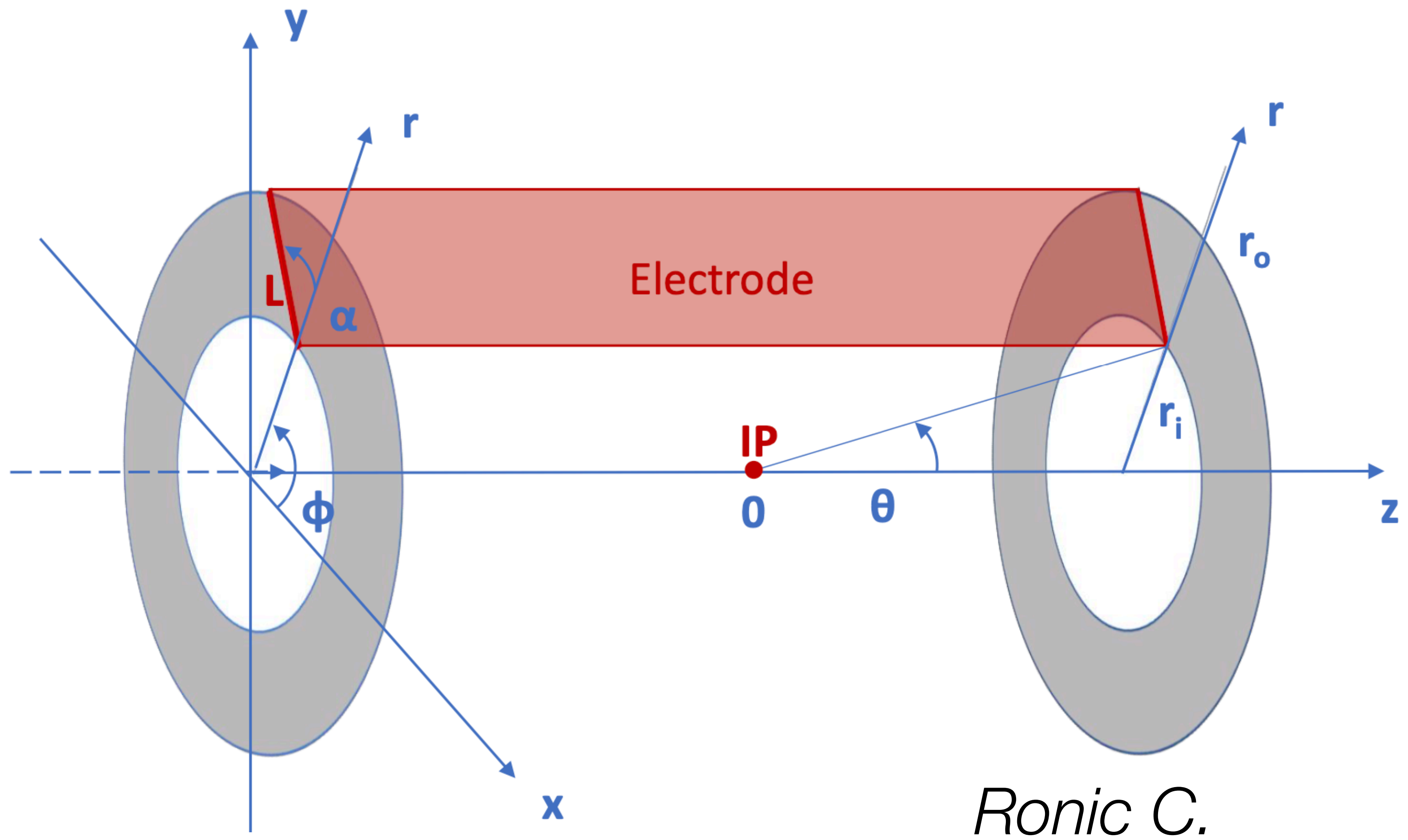
Giovanni Marchiori (APC Paris)

FCC software meeting
25 September 2023

FCC-ee LAr calorimeter concept



Brieuc F.



Ronic C.

- FCC-ee LAr calorimeter concept has a non-trivial geometry with non-projective modules inclined in R-phi and with logical grouping of cells along theta and module direction that can differ among the 12 radial layers of the detector concept
- Previous segmentation/digitisation/clustering classes are not adapted to this geometry & readout, and new classes have been/are being developed (see this presentation and PRs for segmentation/digitisation: https://github.com/HEP-FCC/FCCDetectors/pull/56, https://github.com/HEP-FCC/k4RecCalorimeter/pull/48, https://github.com/HEP-FCC/k4SimGeant4/pull/46)
- To check that merging and positioning of cells in space was working as expected, as well as the clustering of the cells into topological cluster, I wrote a little event display in ROOT to overlay detector geometry/readout/hits/cells/clusters

Event display requirements and chosen framework

- Main wishes:
  - Load directly the geometry used in Geant4 rather than draw it by hand based on the input parameters, to avoid inconsistencies (geometry creation is quite complex)
  - Read event data information stored in ROOT files
  - Automatic 2D projections (R-phi and R-z) from 3D data
  - Small GUI for interactivity with user: enable/disable geometry elements, switch between events
- No familiarity with any specific tool beforehand - decided to use ROOT's graphics classes to write something quickly and do everything inside ROOT, based on the EVE classes
  - Pros:
    - demonstrated to work in e.g. Alice event display
    - documented online on ROOT website
    - some tutorials available, doing things similar to what I was looking for
  - Cons:
    - documentation often limited to class hierarchy/methods + examples and some conference proceedings
    - code is quite old (< 2010) and now only in maintenance mode (no new development)
    - small developer/user base => hard to find solutions online to problems not covered by the tutorial and to understand what happens behind the scenes (BUT: main developer, Matevz Tadel, has been very kind and quick in answering an e-mail from my side asking for help once)

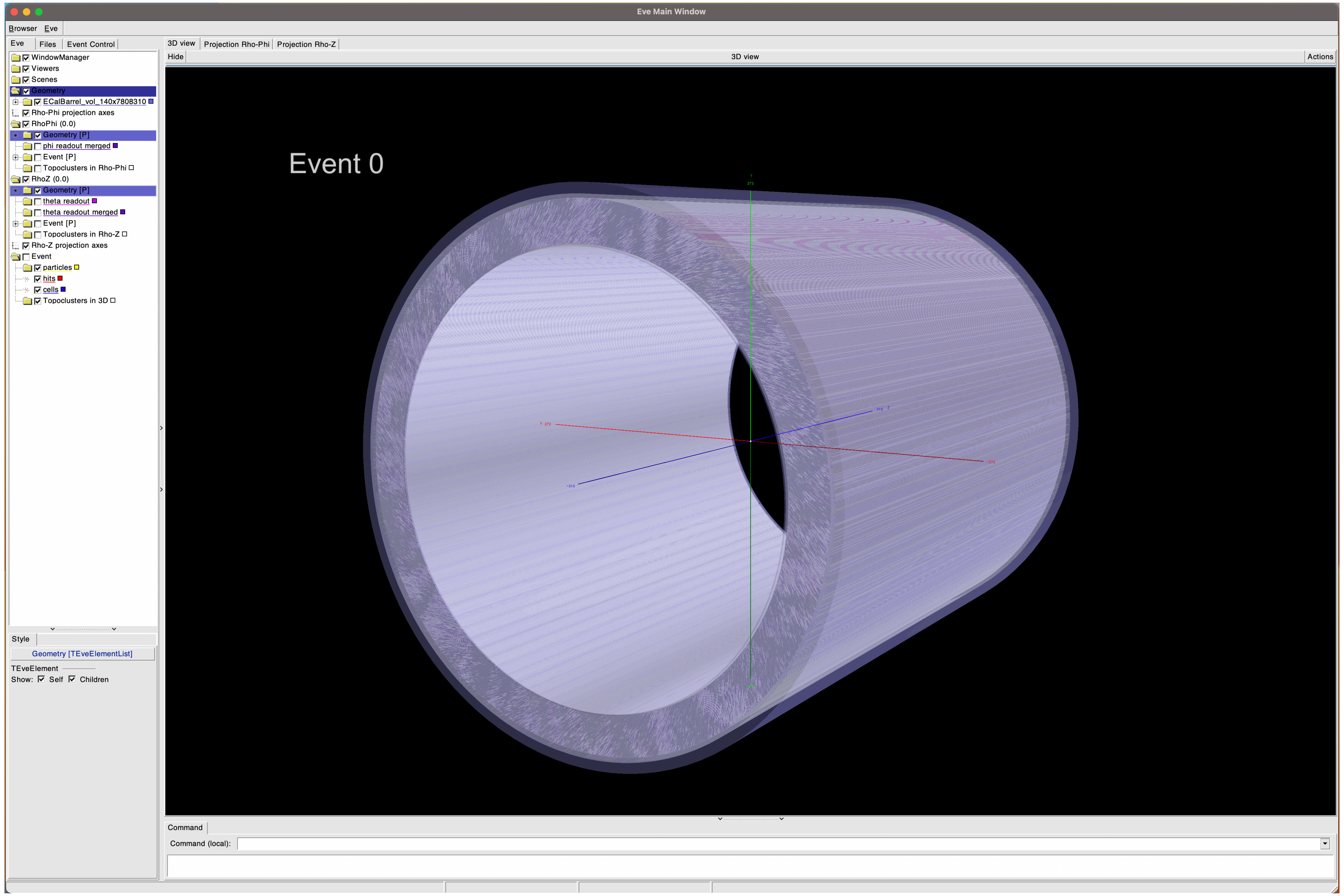
Detector geometry

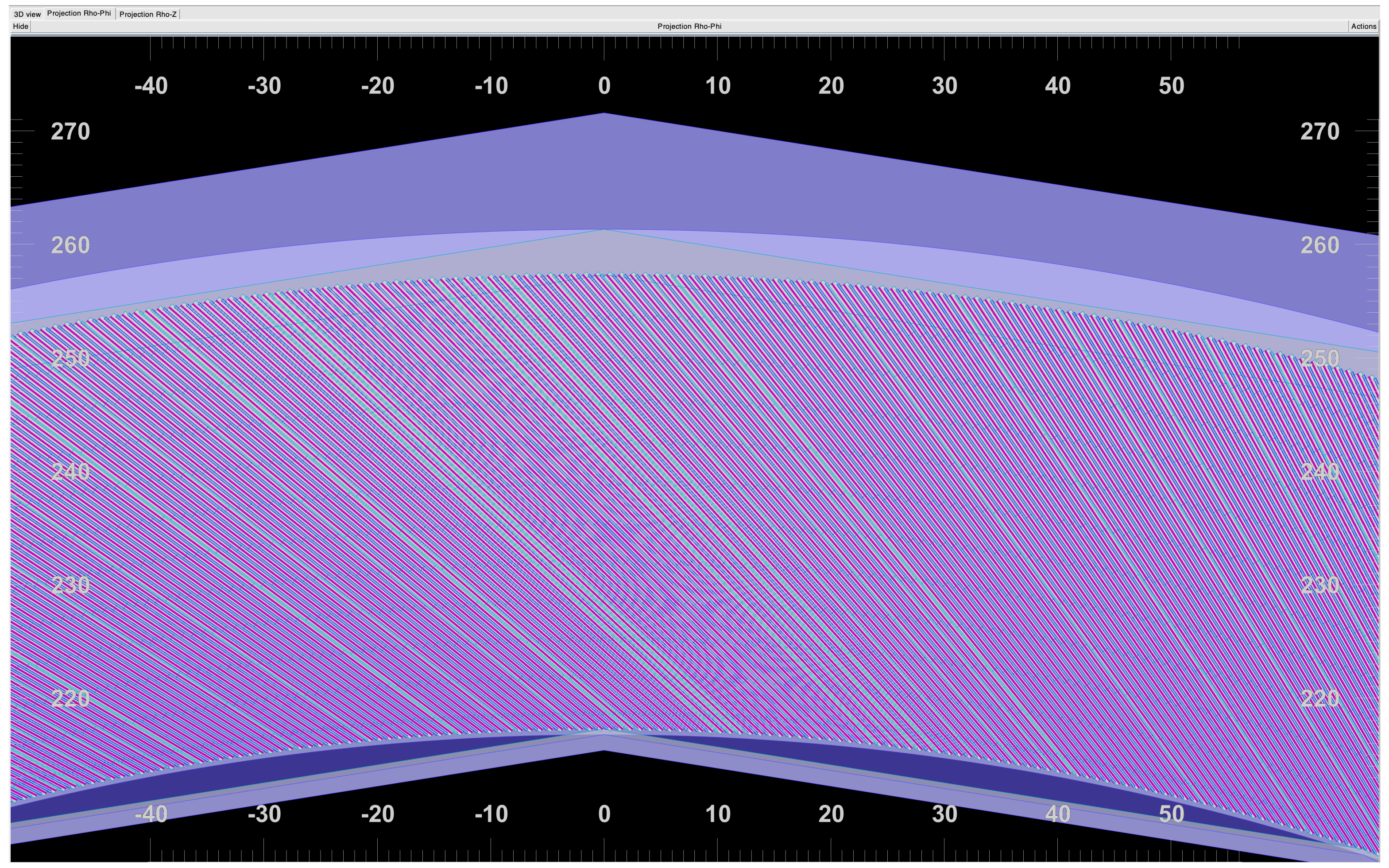
- The detector geometry is loaded and drawn only once at the beginning of the program
- It is represented as a hierarchy of **TEveGeoNode**(s) descending from a **TEveGeoTopNode**
- The model is exported once from fccrun simulation via G4->GDML converter (using **GeoToGdmlDumpSvc**)
- The GDML file is then imported in ROOT and converted to a TGeoVolume via

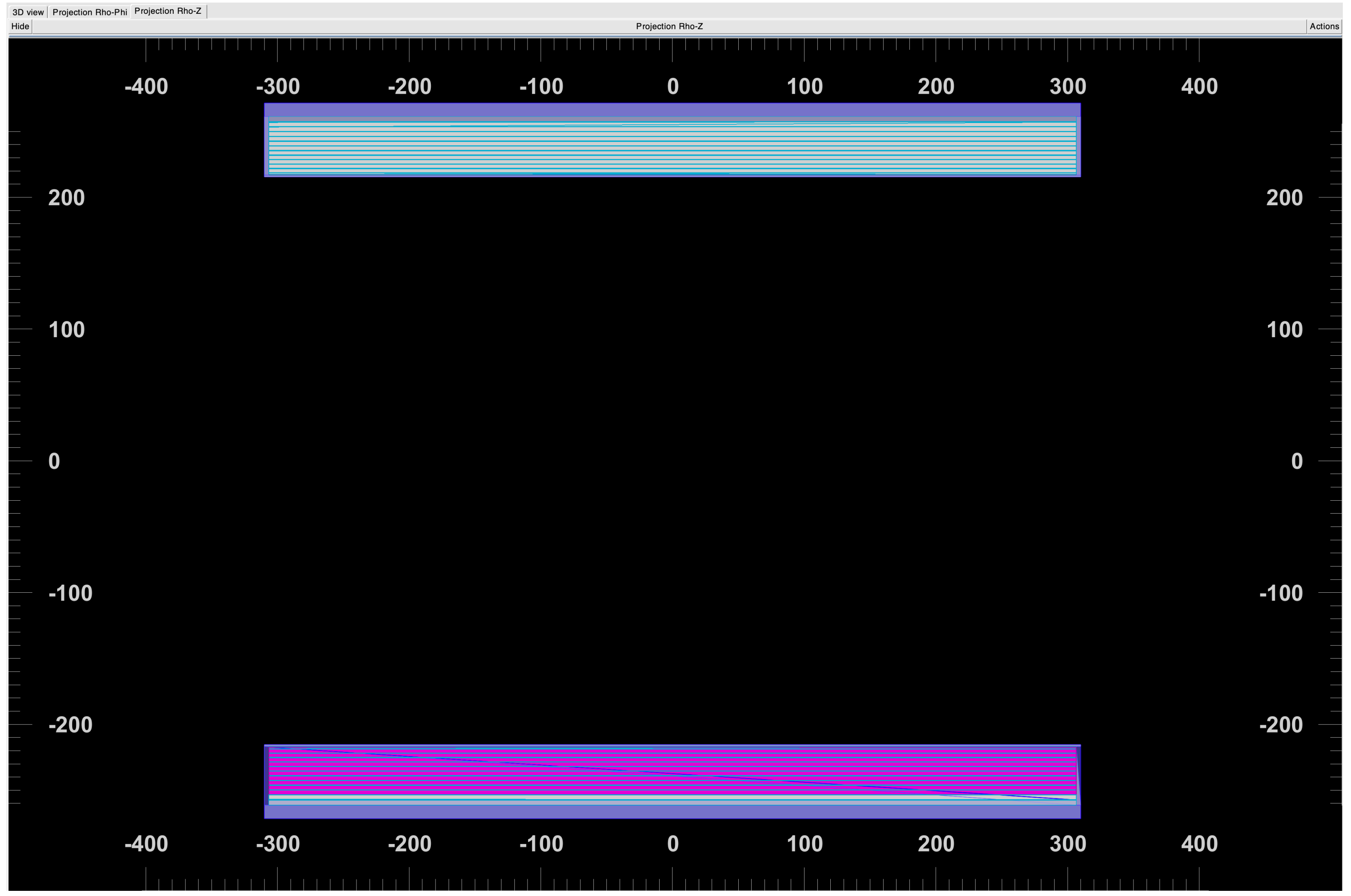
```
gGeoManager = TGeoManager::Import(“file.gdml”)
TGeoVolume* topVol = gGeoManager->GetVolume(“topVolName”);
auto node = gGeoManager->GetTopVolume()->FindNode(“topVolNodeName");
TEveGeoTopNode* inn = new TEveGeoTopNode(gGeoManager, node);
```

- However, some 3D volumes (e.g. Tubes) do not project properly in 2D in EVE
  - Found a workaround:
    - save the TEveGeoTopNode to a ROOT file as a **TEveGeoShapeExtract*** (if I understand correctly this converts various 3D volumes to 3D multi-polygon shapes) with TEveGeoTopNode::SaveExtract(..)
    - Read back the TEveGeoShapeExtract(for the event display (open TFile, use TFile::Get to read the TEveGeoShapeExtract*, and then build the 3D model with TEveGeoShape::ImportShapeExtract(..))
- Once the geometry is built, 2D projections are calculated, views created, and added to the browser on the left side of the window
- As the geometry is very detailed, some elements (the LAr bath and modules) are not drawn by default in the 3D view (slow)

Detector geometry: 3D and 2D views



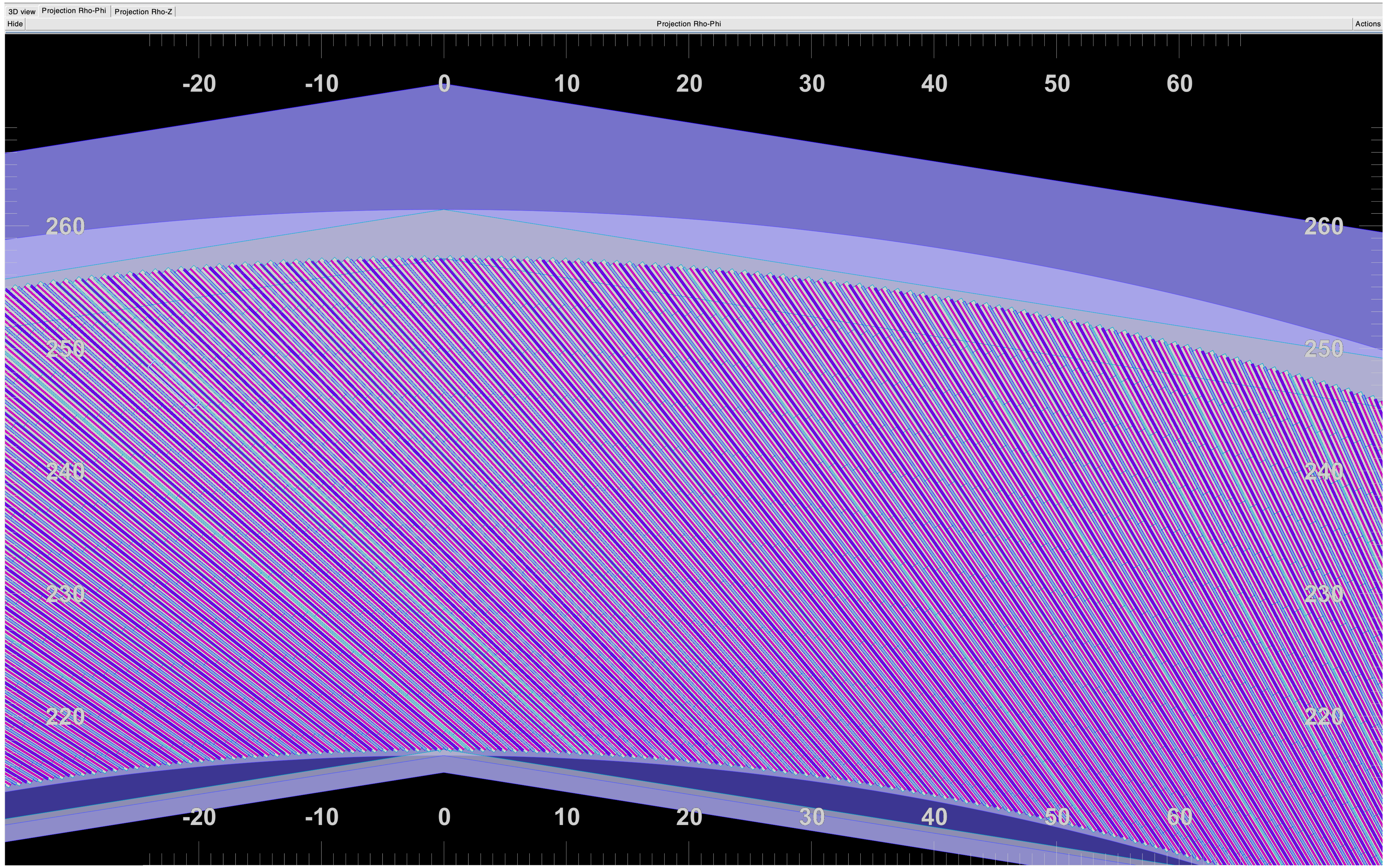


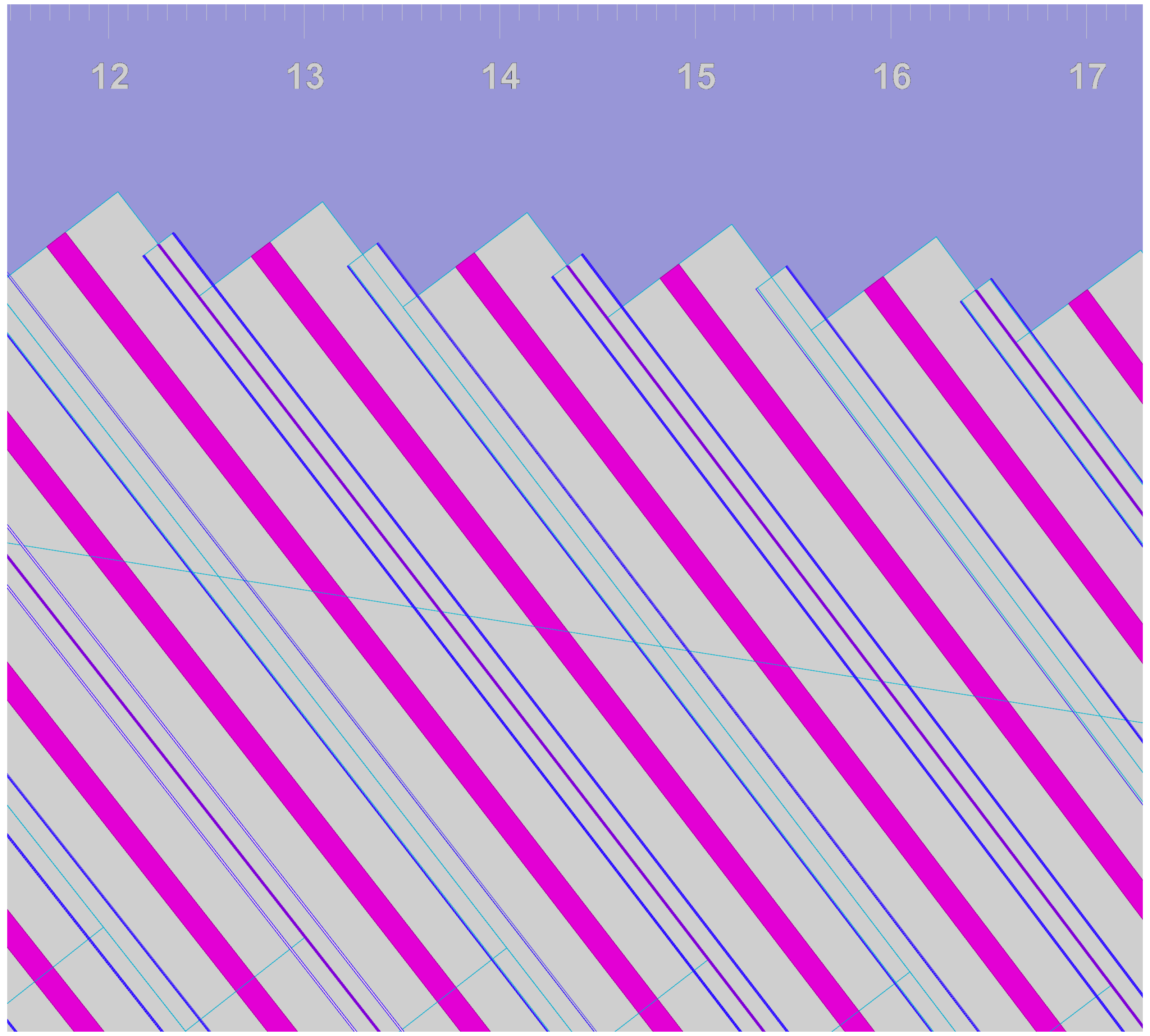


Detector readout

- The readout, like the geometry, is created and plotted once at the beginning of the program
- The detector G4 module is not segmented along theta - the theta readout is only implemented in software at hits/digitisation level
- Similarly, the grouping of cells along theta and/or phi is only done at digitisation level - it's not inherent to the G4 model and thus not persisted with it in the GDML file
- The theta/module readout grid is thus calculated and plotted on top of the geometry as a set of thicker lines - and added to the display
- Currently, rather than a set of 3D volumes that are then projected to the 2D views, it is represented by two sets of straight lines (one for the R-phi view and one for the R-z view), using the **TEveStraightLineSet** class
    - I went directly in this direction thinking it would be easier and would render more quickly, but haven't thought about the pros/cons of using 3D volumes and letting them project automatically nor didn't check whether speed would be a bottleneck in that case

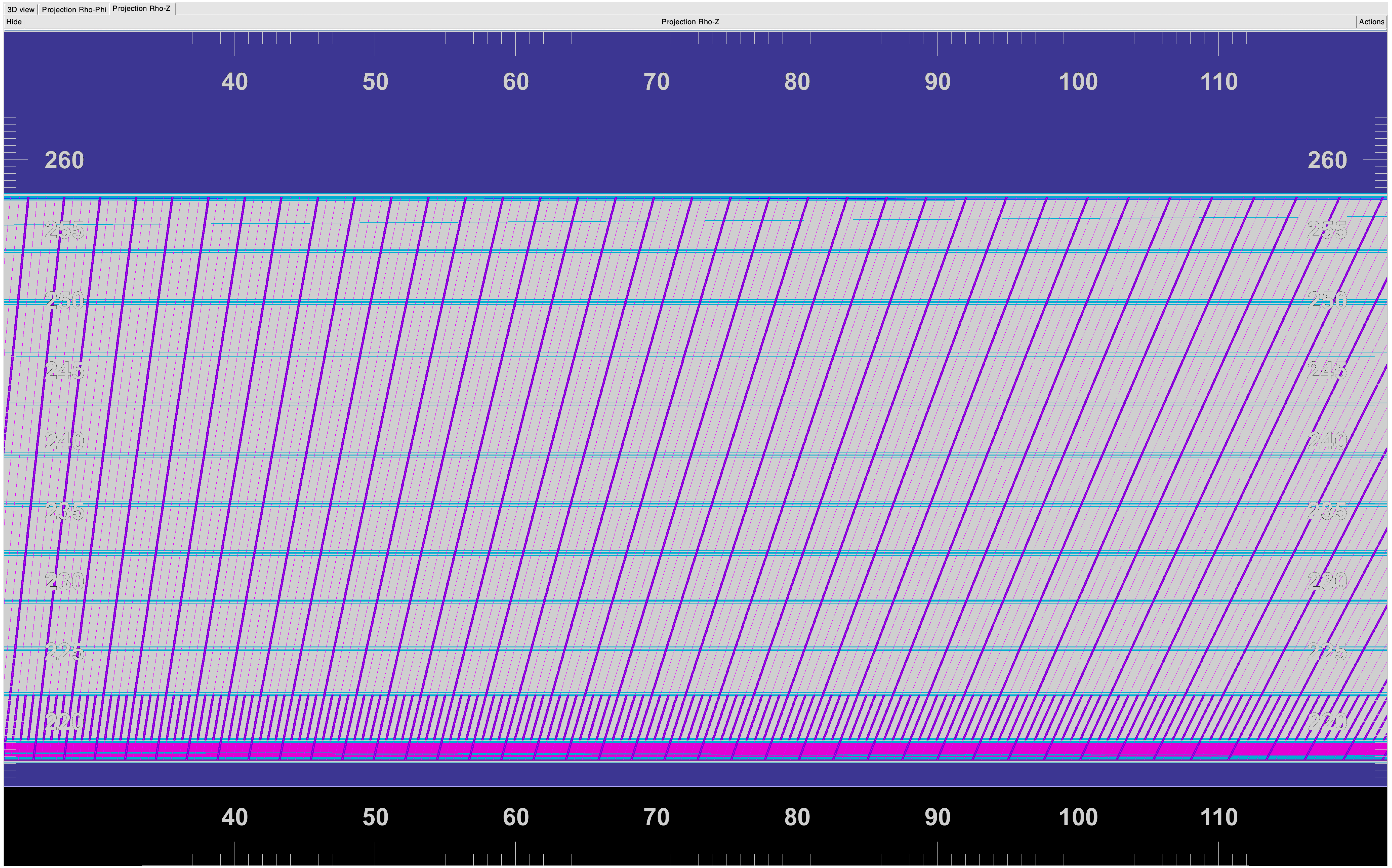
Detector readout: R-phi projection





- Purple rectangles: the PCB
- Blue lines: absorbers (base for unmerged readout)
- Violet lines: the merged readout
- Cyan lines: separations among cells in R and module directions (define G4 volumes)

Detector readout: R-z projection

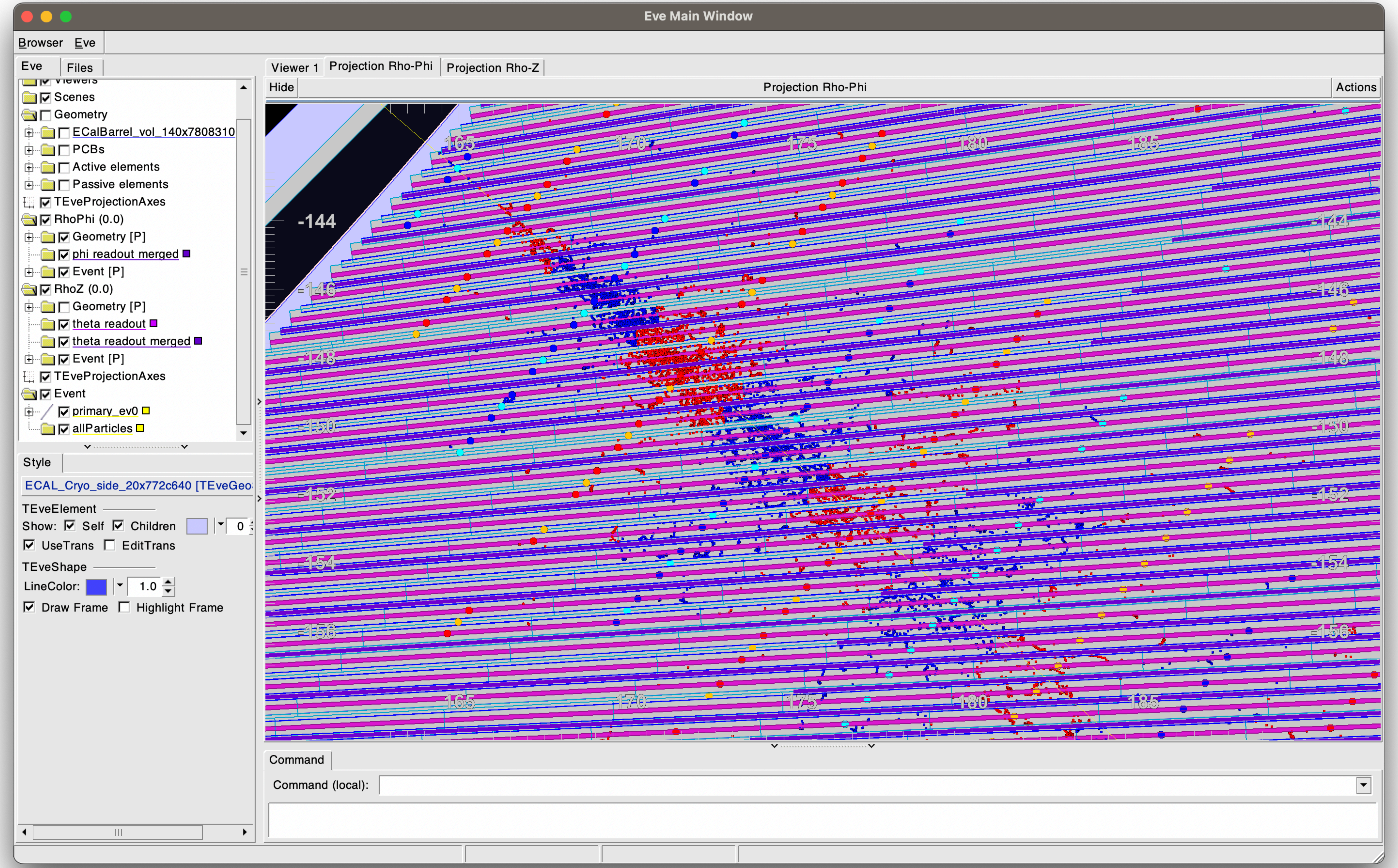


- Thin purple lines: the unmarked readout
- Violet lines: the merged readout
- Cyan lines: separations among cells in R directions (define G4 volumes)

Event data

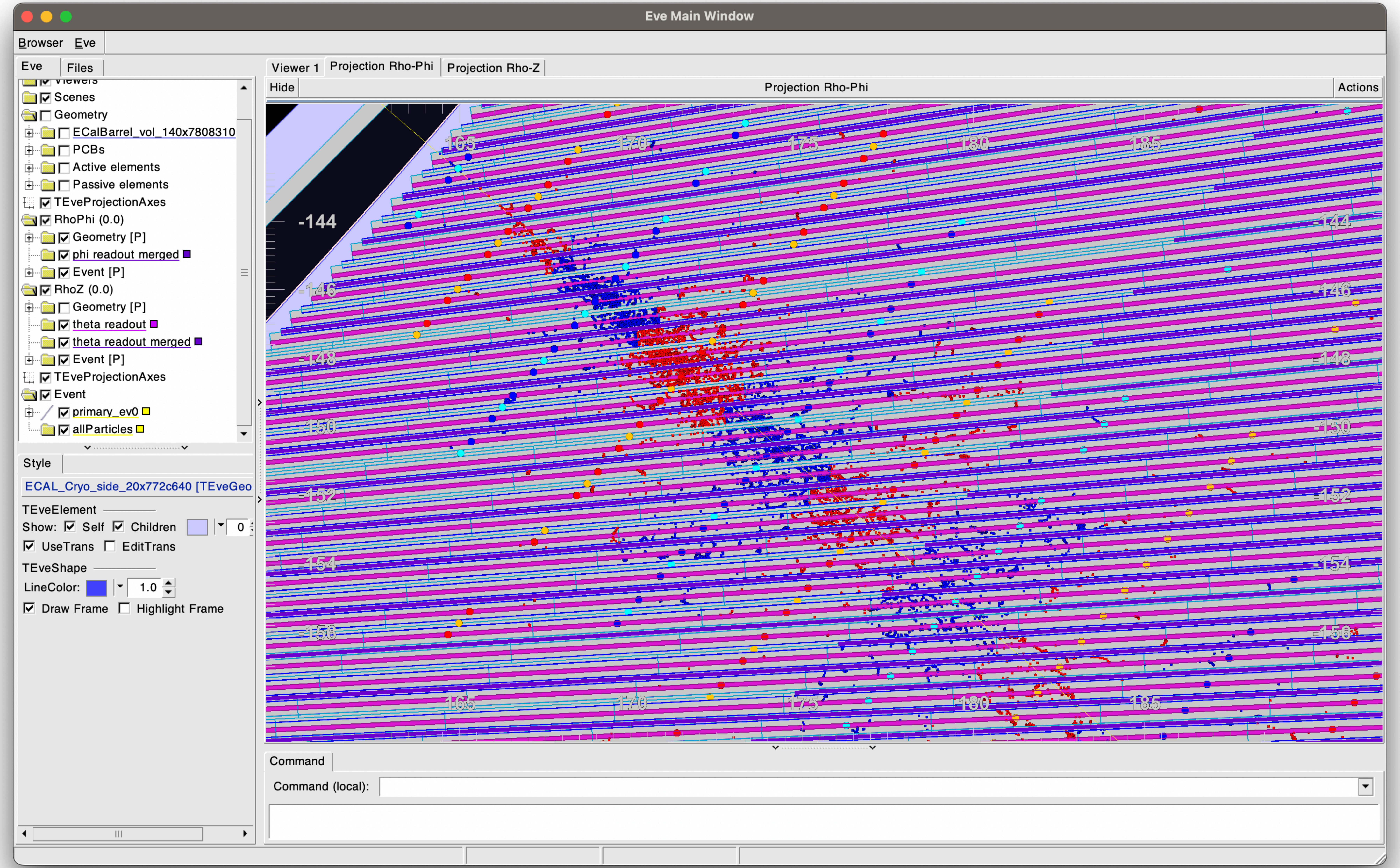
- Event data is loaded and drawn once per event, reading data from an input TTree via **TTreeReader** and filling the in-memory eve objects used for the display
- Event 0 (or another event of choice) is loaded at the beginning, and then user can navigate through forward-backward buttons of the GUI
- Hits and centres of cells are shown as 3D dots, automatically projected also in the 2D views
  - For initial debugging of the per-layer merging of cells along theta/module directions, user could display both cell collections before/after merging, with different colours for even- vs odd-numbered layers
  - Now by default only the cells for the default (merged) readout are shown and with same color for all layers
  - The class **TEvePointSet** is used to record the position information and plot the dots
- TopoClusters (connected topological 3D clusters of cells) are shown in both the 3D views as filled 2D/3D polygons using the **TEveQuadSet** and **TEveBoxSet** classes (TEveBoxSet does not project to 2D..). They hold the vertices of the polygons, a cell ID and an additional channel number that can be used for choosing the color of the cell in a palette (here: cell energy)
  - Note: there are **TEveCaloData / TEveCalo3D / TEveCalo2D** classes that could do the job and project properly but they are designed for (th)eta-phi projective cells so they do not work in this case..
- The G4 generated particle direction is also displayed, as a straight yellow line (no B field so far) using TEveStraightLineSet
  - Tried so far unsuccessfully to use TEveTrackList, which would make it possible to (1) display more information about the particle (PDG code, ..) and (2) draw curved **trajectories** for charged particles if B!=0

Event data: hits and cells (old version)



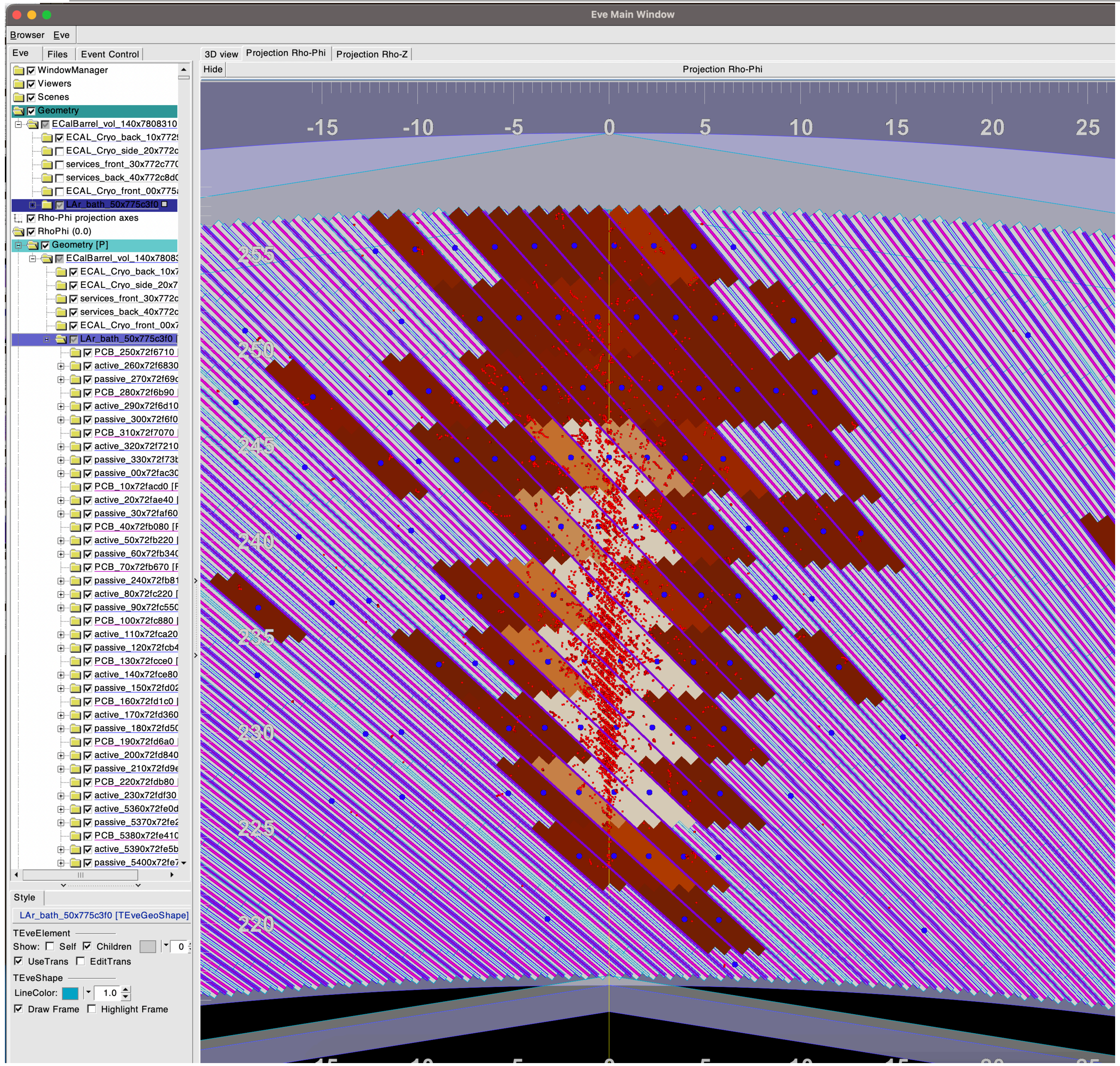
- In this test, merged cells per layer= 2,2,2,2,2,2,1,1,1,1,1,1
- Yellow line: the primary e-
- Blue/red small dots: the hits in even/odd-numbered layers (from hit cellID)
- Blue/red circles: the cells in even/odd-numbered layers before module merging
- Cyan/orange circles: the cells after merging (in this case, N=2 in inner 6 layers and N=1 in outer 6 layers)

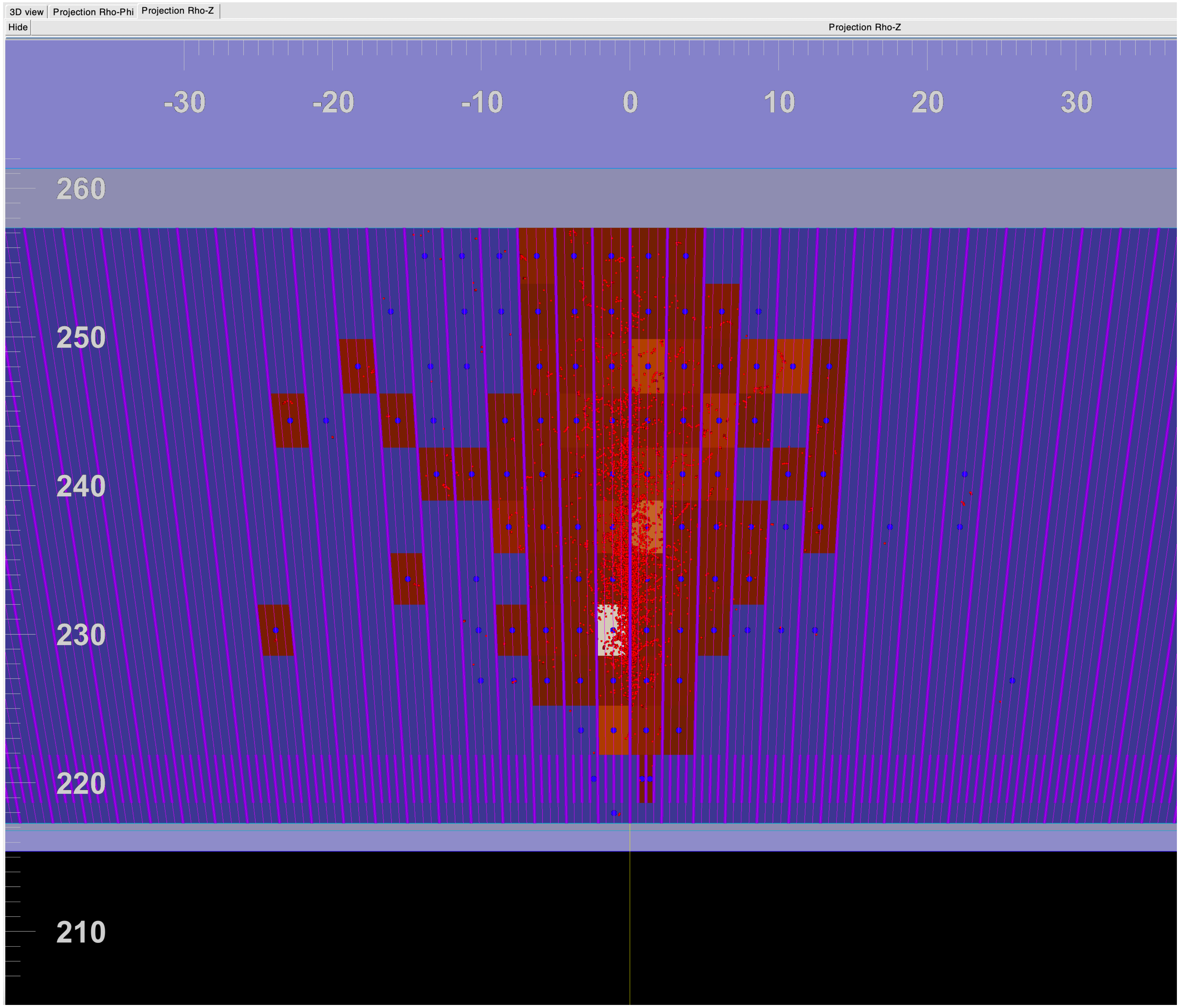
Event data: hits and cells (old version)



- In this test, merged cells per layer = 4, 8, 2, 1, 8, 4, 2, 1, 4, 2, 1, 8
- Same color code as before, now in Rho-Z projection

Event data (current version)

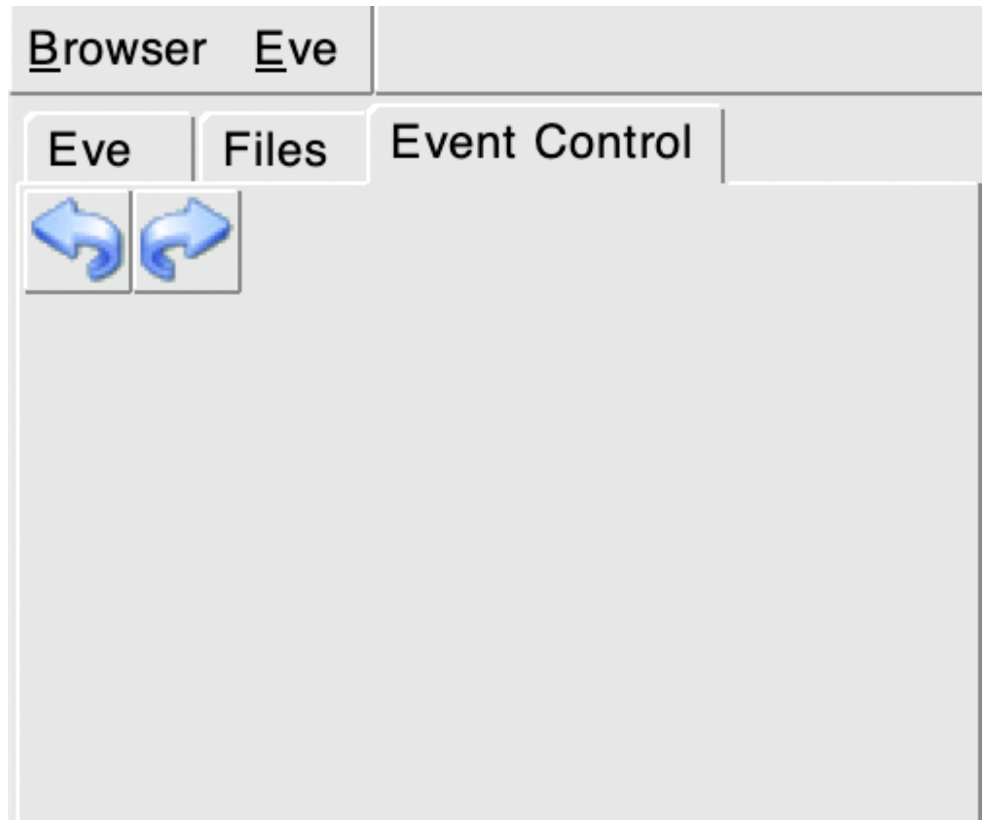


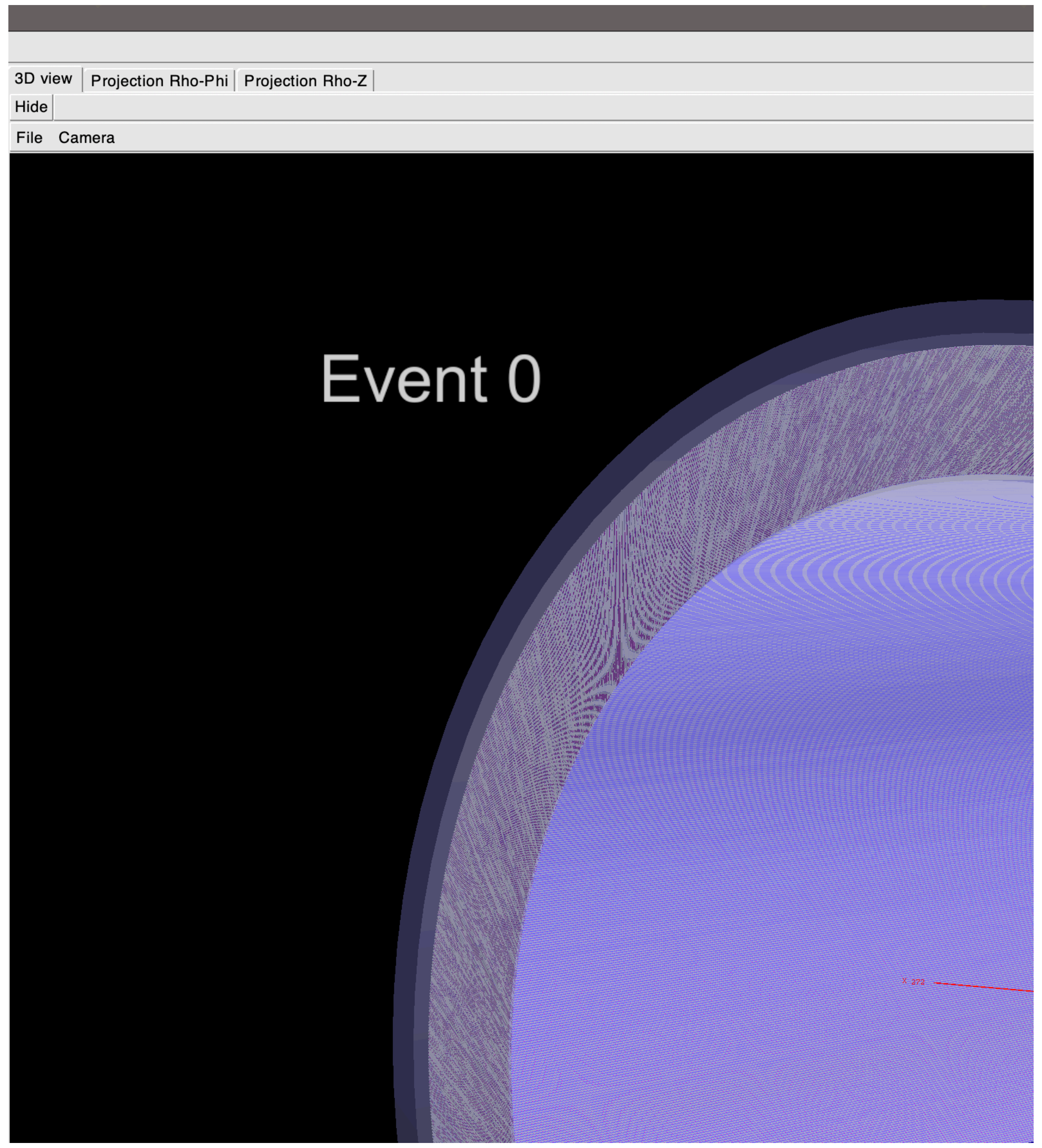


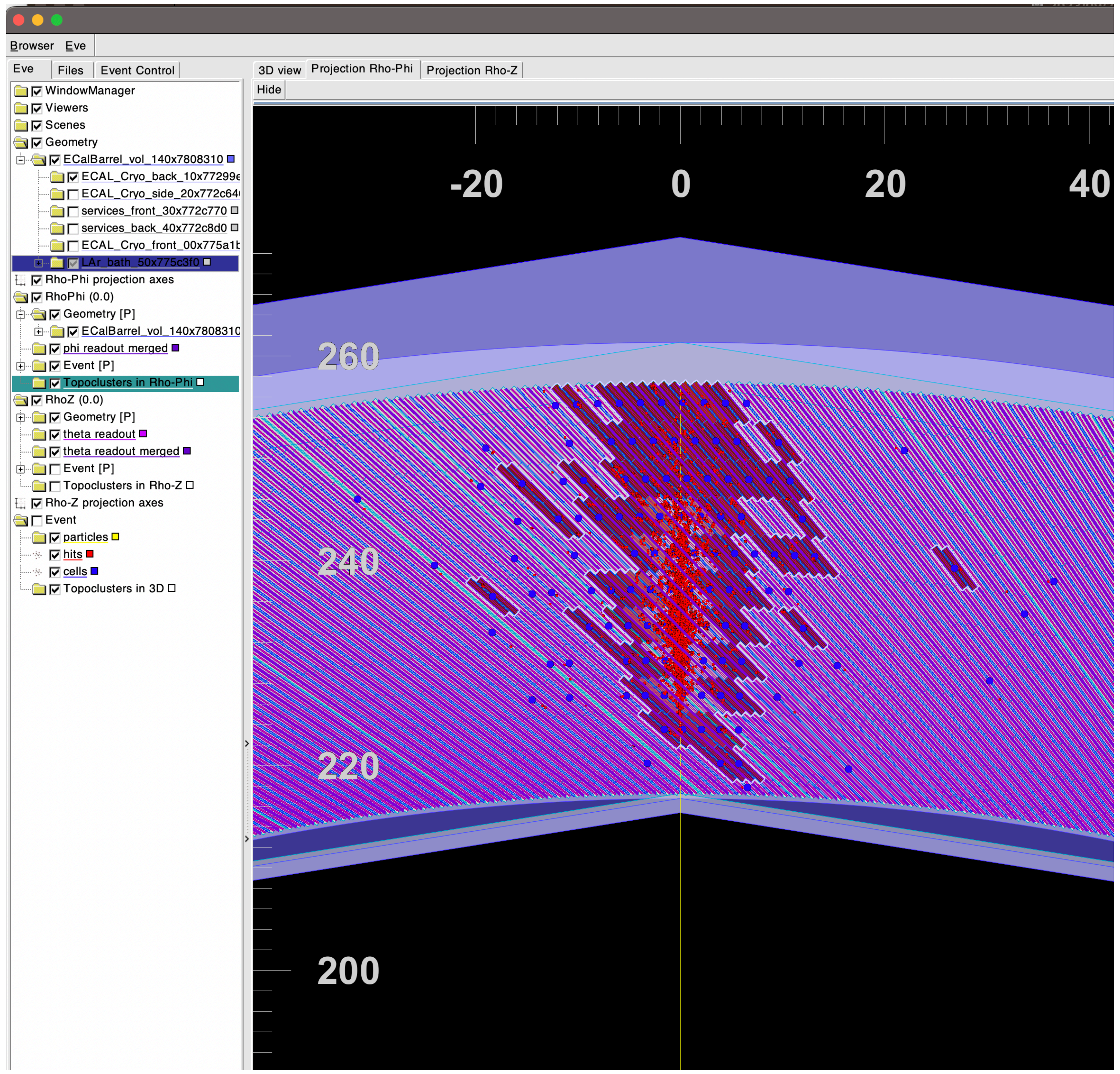
- Hits are shown in red
- Only default cells (blue dots) are shown, here with 2x merging in modules direction and x4 merging in theta in the 2nd longitudinal layer
- Cells belonging to topoclusters are filled (brown shades, lighter = E↑)

GUI (current version)

- 3rd tab in left panel ("Event Control") allows navigation through events using **TGPictureButtons** linked (via ::Connect) to functions that increase/decrease the ID of the event to be displayed and then call the function that loads and draws data for the current event
- Current event number is displayed in 3D view via **TGLAnnotation**
- Display is interactive - one can click on objects and see them highlighted in the browser and viceversa







Some stats

- Code is ~1.1k lines long
    - ~250 lines of includes, constants, helper functions (mainly to deal with geometrical calculations for the readout)
    - ~100 lines for GUI-related stuff
    - ~500 for class that reads events and fills the in-memory objects for the event graphical representation
    - ~250 lines for main function setting up the elements of the main window, loading the geometry and displaying the geometry and readout, opening the event ROOT file and calling the event reader on the first event

Open issues and conclusions

- Open issues:
  - Plotting order sometimes not easy to control (calculated automatically based on diagonal of objects) - some objects (e.g. clusters) can hide others (e.g. cell positions)
  - Overlaying palette for cluster color scale for some reason leads to slow rendering, cause not yet identified -> hidden for the time being
  - Has not been able to solve issue with using TEveTrackList, get weird error about locked objects
  - …

- Despite little issues here and there, the event display serves the purpose I wrote it for, making it possible to visualise the output of code modifications and thus rendering debugging a bit easier
- The code was written quite quickly and could certainly be improved…
- Latest version: https://github.com/giovannimarchiori/LAr_scripts/blob/gmarchio-main-2023-09-08-newclustering/FCCSW_ecal/displayV3.C
- Run with:
  - root
  - .L displayV3.C+
  - display()