The next generation of

OpenGL support in ROOT

Matevž Tadel

Including work from: Alja Mrak-Tadel & Timur Pocheptsov

Contents

Status @ CHEP-06

Work done by R. Maunder & T. Pocheptsov in '05

Based on `TVirtualViewer3D` API

Use `TBuffer3D` for all transfer of data to viewer

Impressive features:

- Optimized for geometry rendering, support CSG operations
- Support clipping / view frustum culling
- Support view-dependent level-of-detail

Issues when used for ALICE event-display:

- Scene-updates drop all internal state ➔

  Not suitable for frequent refreshes / small changes
- Hard to extend for classes that require complex visual representation (e.g. raw-data)

  But this was a known trade-off for using `TBuffer3D`.
- Stand-alone viewer victim of feature pile-up

  Difficult to add new features or even extend existing ones.

Evolution of OpenGL support

Jan-Aug '05: explore GL on ALICE Pb-Pb events

60k tracks, 10M TPC hits → too much data → **interactivity is the key**

Early '06: prototype of ALICE display using ROOT GUI & GL

Apr '06: direct OpenGL rendering for ROOT classes

Aug '06: two-level selection (pick container contents)

Accumulation of issues → reflection break ➔ **Manifest:**

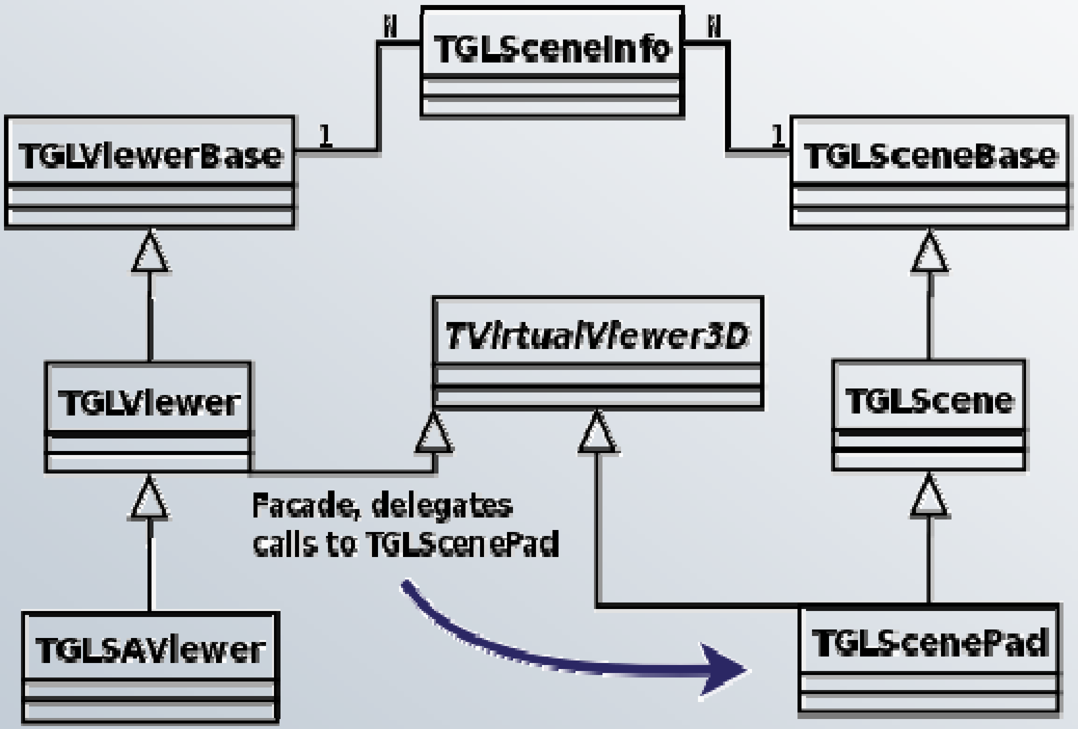
I. **GL becomes the main 3D engine** – minimal support for others

II. **Gradually restructure GL to achieve the following:**

1. Support multi-view displays with shared scenes
2. Optimize update behavior for dynamic scenes
3. Display 2D graphics primitives in GL
4. Include external GL engines in ROOT viewer
5. Include ROOT scenes in other environments / toolkits

Jul `07: most of the above done in ROOT 5.16 production release

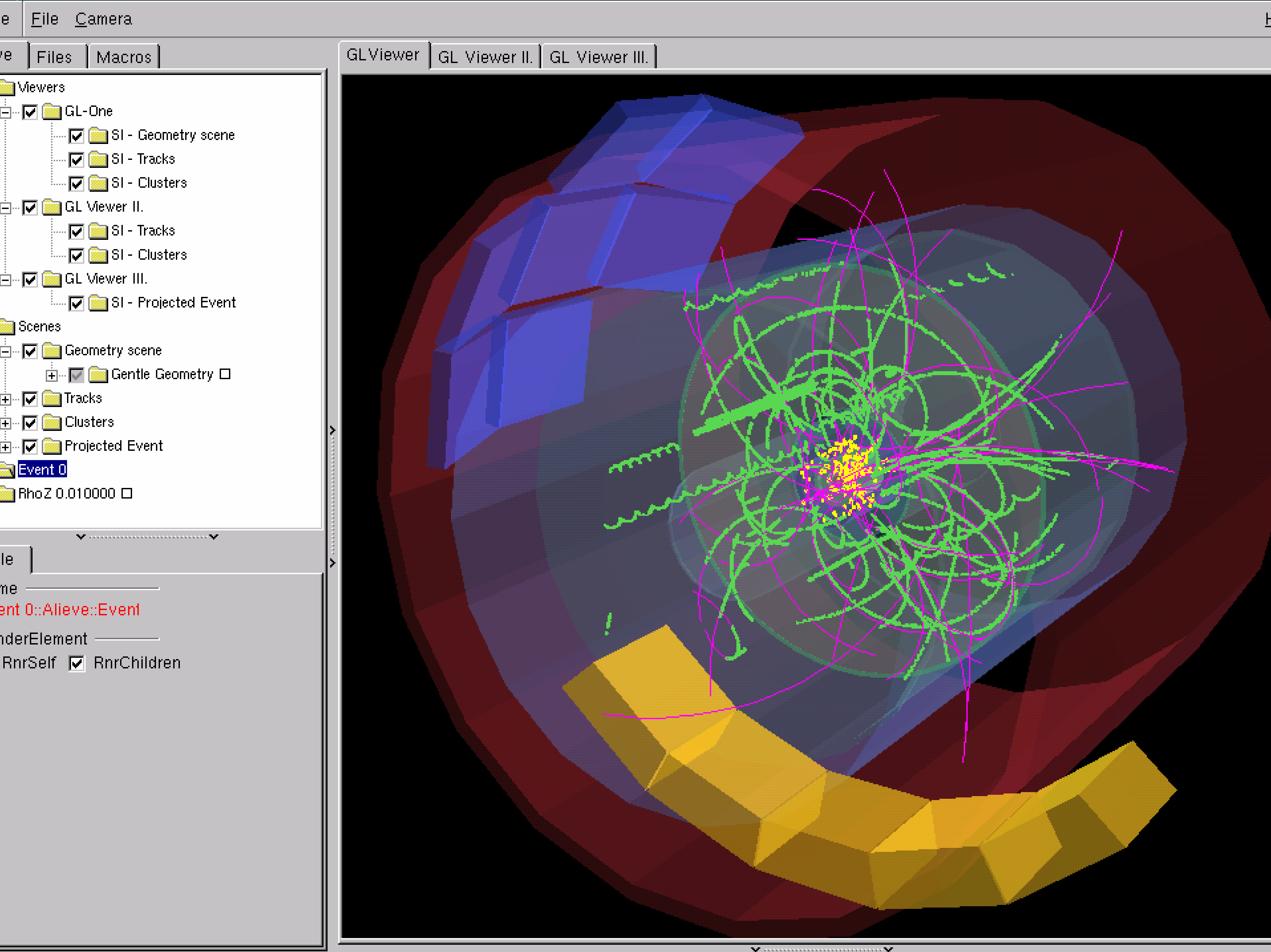
New Viewer—Scene diagram



- **TGLSceneBase**
  Bounding-box ➔ draw visible only
  Viewer-list ➔ updates
  Place to plug-in foreign scenes
  No assumptions about content
- **TGLScene**
  Containers for logical/physicals shapes
  Cleaned version of old scene
  Supports fine-grained updates
  Use this to 'export' a ROOT scene
- **TGLScenePad**
  Natural inclusion of pad-contents:
  thus we can service old classes!
  Notice VirtualViewer3D inheritance.

- **TGLSceneInfo:** "scene-in-a-viewer", caches view-dependent information
- **TGLViewerBase**: minimal; a collection of scenes + render steering
- **TGLViewer**: adds selection interface & event handling (already ROOT specific!)
- **TGLSAViewer**: top-level, stand-alone viewer with GUI

This was **A LOT** of work … but now it's done right!


File
Camera
Files
Macros
GL Viewer
GL Viewer II.
GL Viewer III.
Viewers
GL-One
SI - Geometry scene
SI - Tracks
SI - Clusters
GL Viewer II.
SI - Tracks
SI - Clusters
GL Viewer III.
SI - Projected Event
Scenes
Geometry scene
Gentle Geometry
Tracks
Clusters
Projected Event
Event 0
RhoZ 0.010000
ent 0::Alieve::Event
nderElement
RnrSelf
RnrChildren

Direct OpenGL rendering – I.

Manually implement class for GL rendering, eg:

1. For class `PointSet` implement:

```
class PointSetGL : public TGLObject
{
  virtual Bool_t SetModel(TObject* obj);
  virtual void   DirectDraw(TGLRnrCtx& ctx);
};
```

- In `SetModel()` check if *obj* is of the right class and store it somewhere (data-member in `TObjectGL`)

  The GL object can access data of its creator!

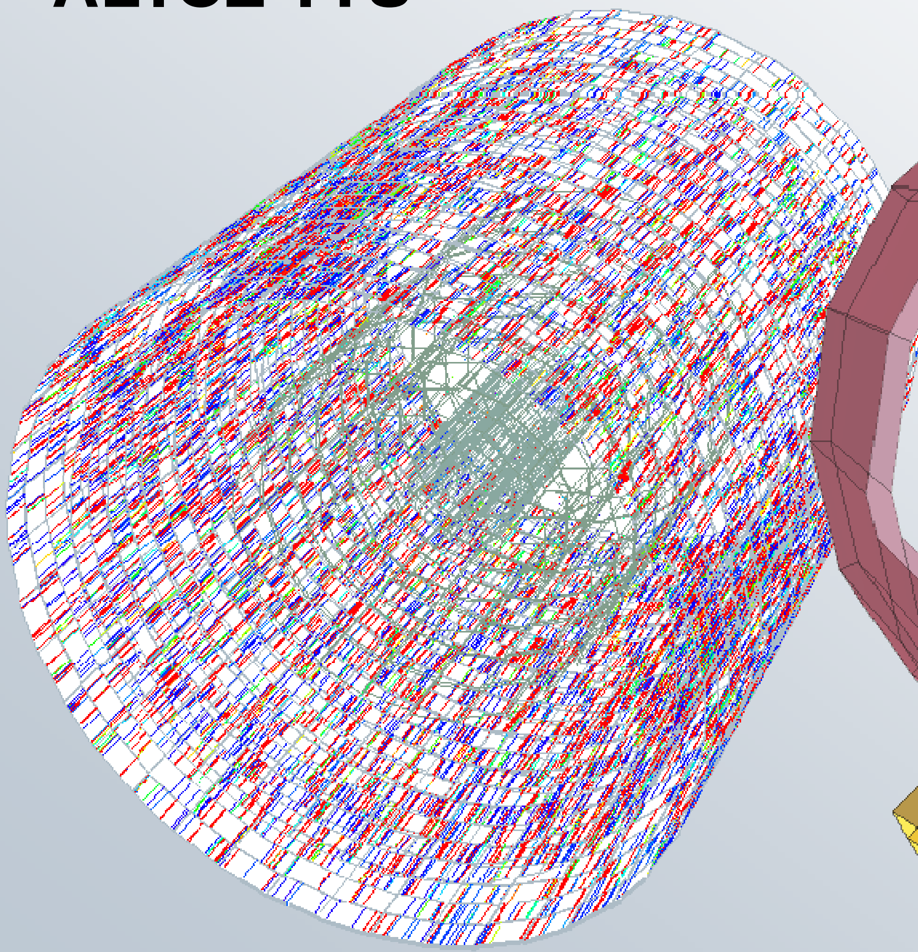
1. `DirectDraw()` is called by viewer during draw-pass

   Here do direct GL calls, change state, draw whatever.

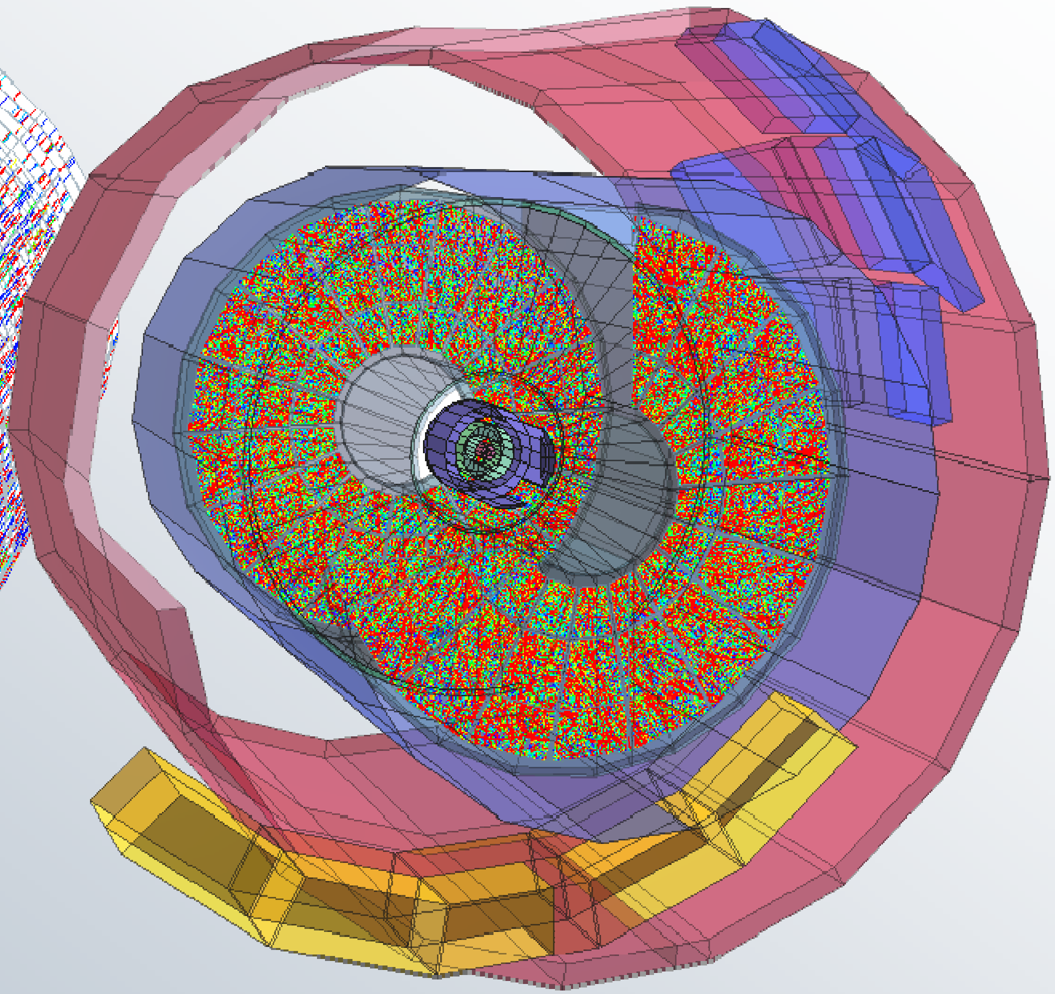
   Leave GL in a reasonable state – others depend on it.

Direct OpenGL rendering – II.

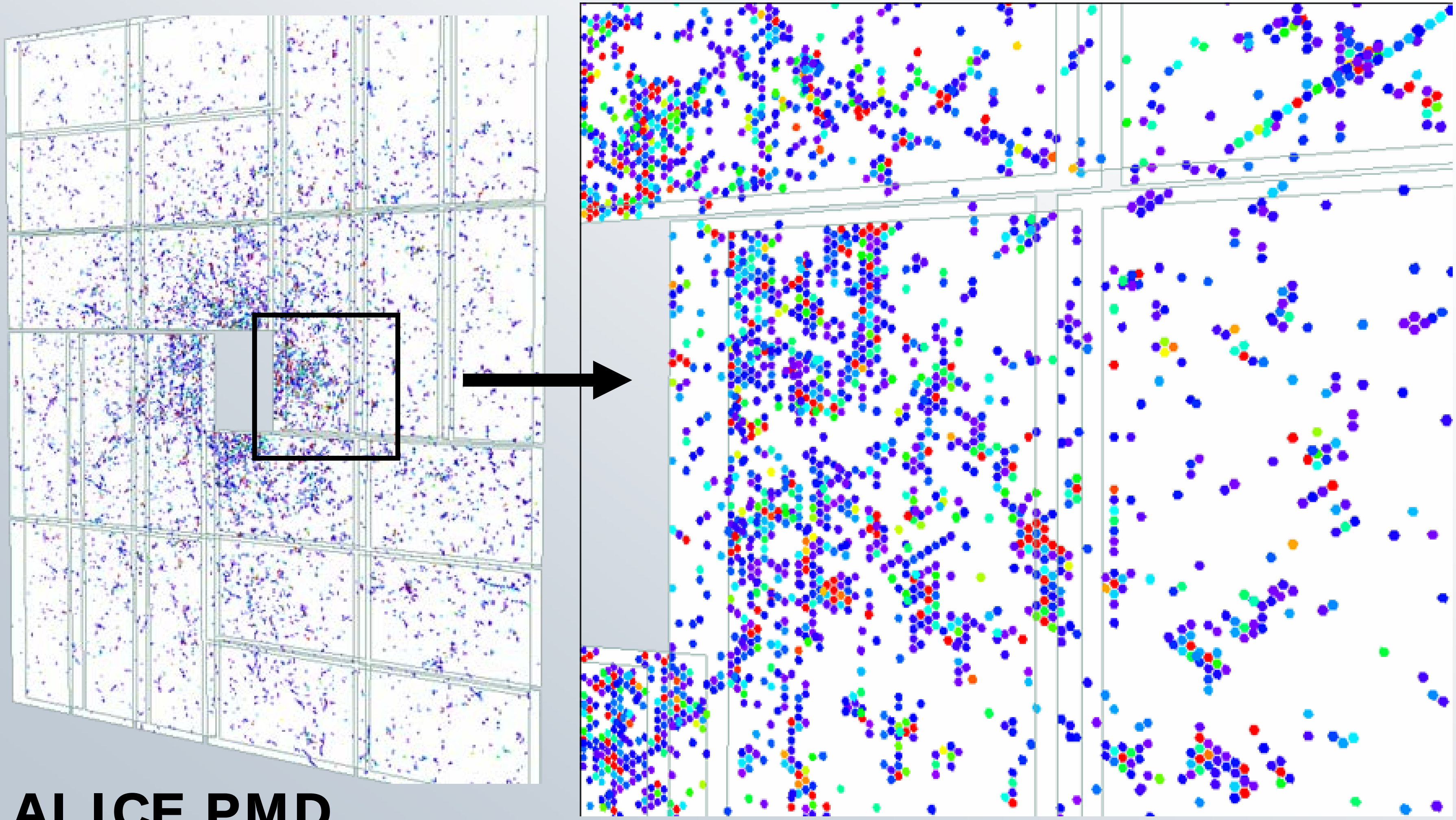
ALICE ITS



ALICE TPC



Direct OpenGL rendering – III.



ALICE PMD

Direct OpenGL rendering – IV.

How this works:

1. In `Paint()` fill only *Core* section of *TBuffer3D*:

   TObject fID*, color, transformation matrix

   Pass it on to viewer.

2. Viewer scans *fID->IsA()* and parent classes searching for *<class-name>GL* class.

   Only once per class ... cache result in a map.

Benefits:

1. Flexibility – users can draw anything

   Not limited to shapes representable by `TBuffer3D`.

   Provide GL-class, everything works with std ROOT!

   A lot can be done with a small number of classes.

2. Avoid copying of data twice (into/from buff-3d)

   Important for large objects (10M hits in ALICE TPC).

Two-level selection – I.

Imagine a list of clusters, array of digits, ...

One would like to:

a) Treat them as a collection
   - Select, move, turn on/off, change color, cuts, ...

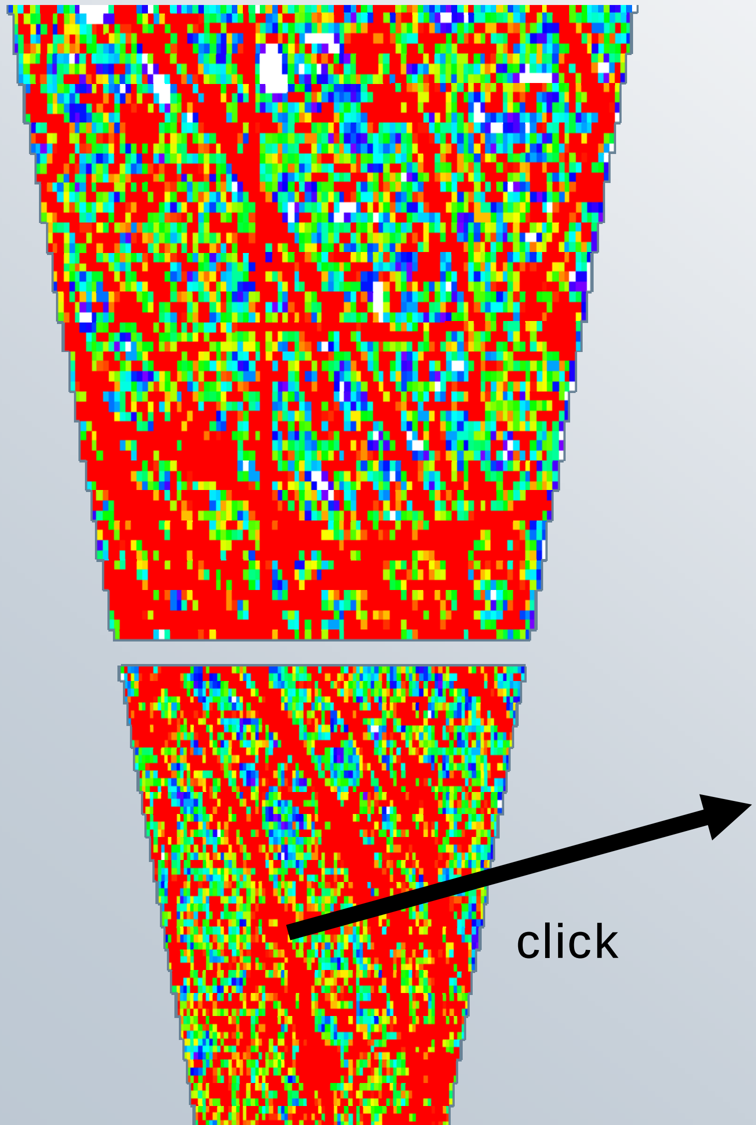
b) Obtain information on individual element
   - Investigate, select for further manipulation

Each element a viewer-object: waste memory/speed

GL supports bunch-processing commands that can not be used in low-level selection mode. Thus use:

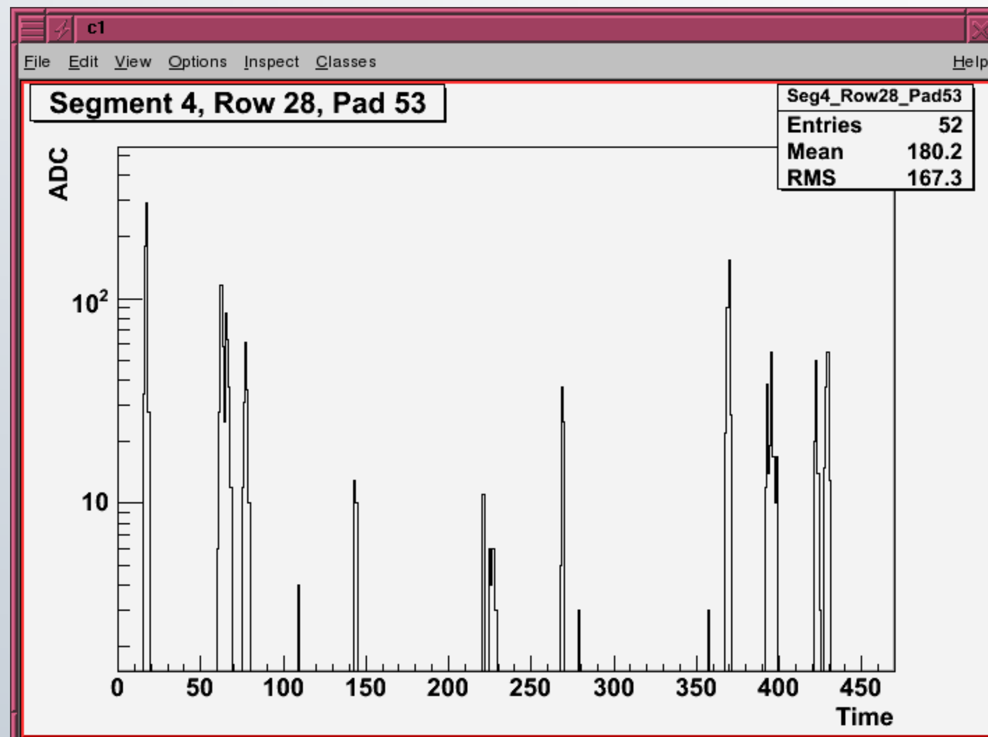
- Optimized version in drawing / first-pass selection
- Special render-path during second-pass (single object!)

Two-level selection – II.



ALICE TPC Sector

1. First-pass: 3 textured rectangles
   Identify object by sector id.
2. Second-pass: ~8000 cells
   Identified row / pad.



Two-level selection – III.

Work is done by the viewer and the GL object:

```
class TPointSet3D : public TGLObject
{
  virtual Bool_t SupportsSecondarySelect();
  virtual void   ProcessSelection( TGLSelectRecord& rec);
};
```

1. First-pass – determine closest object
2. Second-pass – render that object with sub-ids
   The renderer is informed that we're in sec-selection
3. Deliver the selection record back to GL object!
   It tagged elements and should interpret the ids.
   Call function in the master object.
   E.g. TPC row/pad → data-holder can produce histogram

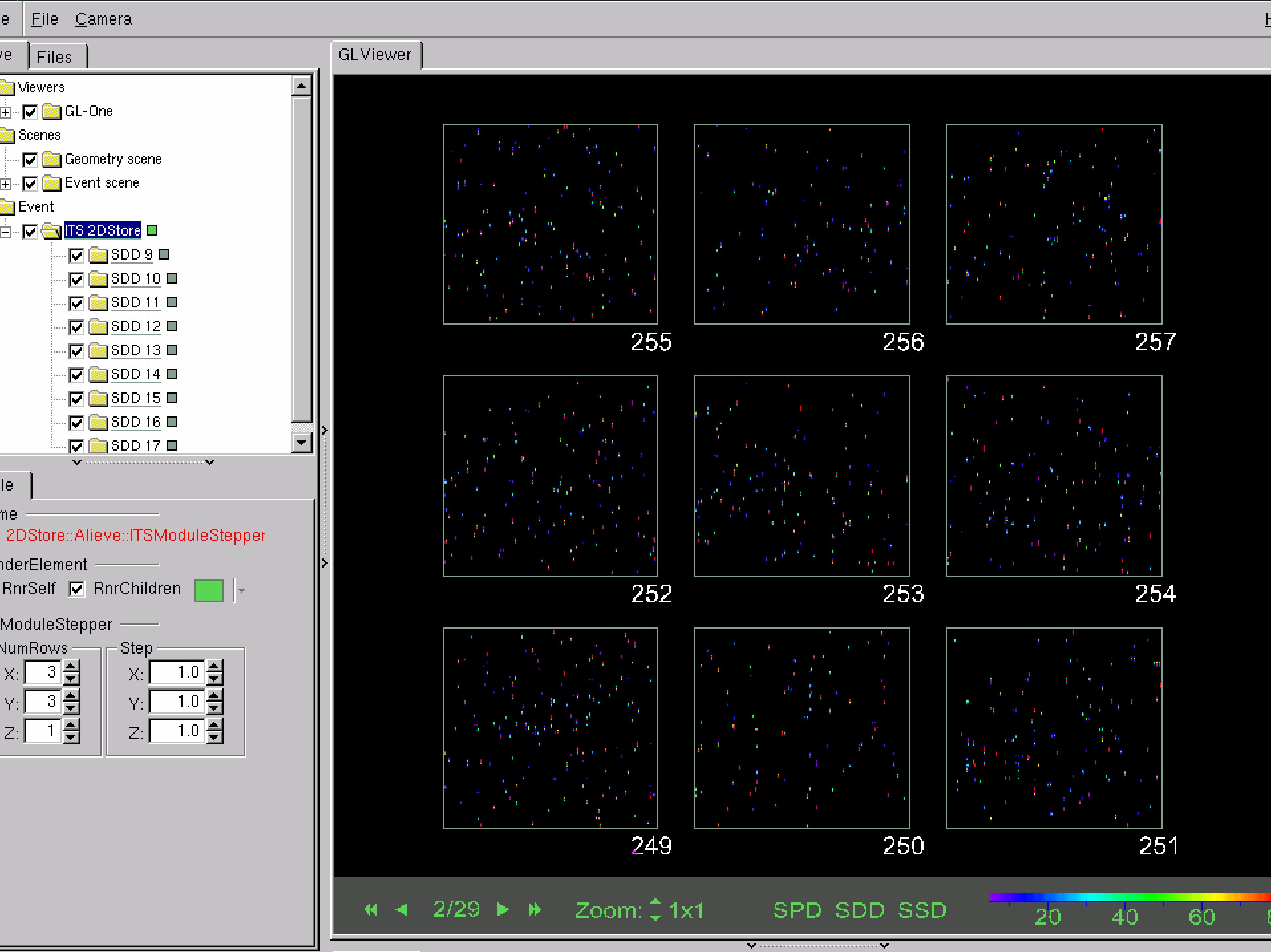
Overlay event-handling

Overlay: a set of viewer-objects that are checked for user interaction on each mouse-move.

Usage:

1. Interaction with objects & dynamic visualization
    1. clipping plane control, object manipulators
    2. modify object parameters that influence rendering
2. Implementation of GUI within GL window

```
class TGLOverlayElement
{
  virtual Bool_t MouseEnter(…);
  virtual Bool_t Handle(Event_t* event, …); // All events!
  virtual void   MouseLeave();
  virtual void   Render(…);
};
```


File Camera
Files
GLViewer
Viewers
GL-One
Scenes
Geometry scene
Event scene
Event
ITS 2DStore
SDD 9
SDD 10
SDD 11
SDD 12
SDD 13
SDD 14
SDD 15
SDD 16
SDD 17
2DStore::Alieve::ITSModuleStepper
nderElement
RnrSelf RnrChildren
ModuleStepper
NumRows
Step
X: 3
Y: 3
Z: 1
X: 1.0
Y: 1.0
Z: 1.0
255
256
257
252
253
254
249
250
251
2/29
Zoom: 1x1
SPD SDD SSD
20
40
60

Pad graphics in GL

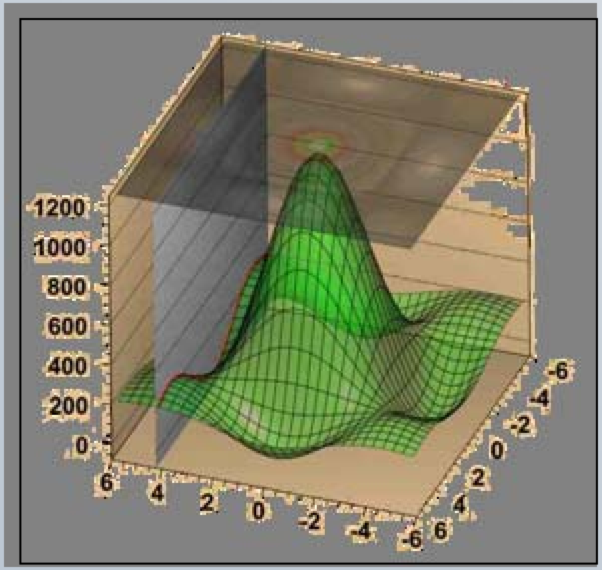
Allow mixing of 3D graphics with:

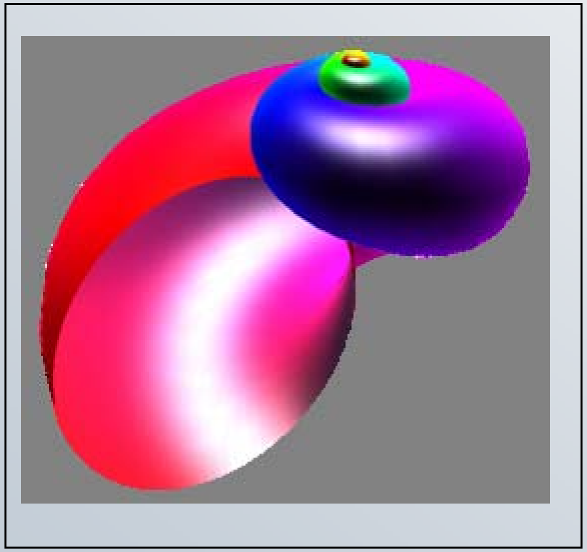
- 2D and 3D histograms
- standard 2D primitives (not done yet)

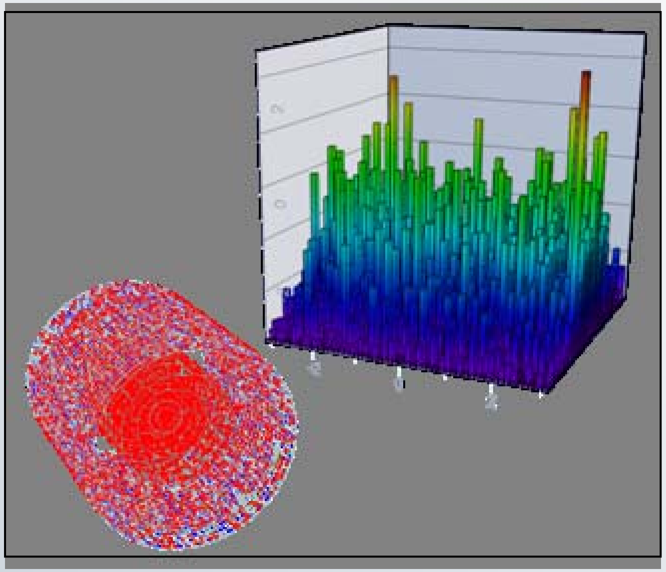
Combine specific event-data with statistical info

2D plots in GL done by *T. Pocheptsov*

- 2 & 3D histograms and functions
- parametric 2D surfaces







Conclusion

- We've made an OpenGL quantum jump
  - Modularization, better control on all levels
  - Overhead-free scene updates
- Development for now driven by the needs of ALICE event visualization framework
  - That's good ➔ heavy-ion events are BIG
  - Interactivity & flexibility
- Experiment-independent part of ALICE event-display will (soon) become a ROOT module
  - The new functionality will become fully exposed