3D graphics with OpenGL

recent improvements and plans

Matevž Tadel

Including work from: Alja Mrak-Tadel & Timur Pocheptsov

Contents

1. Overview of status at ROOT-05
2. Extensions of existing viewer (for ALICE event-display)

   Speed ... flexibility ... interactivity

   Implementation details here ...

   demo with many examples during my next talk
3. Desire for complete OpenGL support

   Started December ’06

   - What has been done already
   - What will be done before summer
4. Conclusion

Status @ ROOT-05 / CHEP-06 – I.

Work done by R. Maunder & T. Pocheptsov

Based on `TVirtualViewer3D` API

- `TGLViewer` just one of 3D viewers; draw via `TPad::Paint()`

Use `TBuffer3D` for all transfer of data to viewer

- classes know their 3D representation, but don't care who renders it and how

Positive (and impressive):

- Optimized for geometry rendering
- Support clipping / view frustum culling
- Support view-dependent level-of-detail
- Support CSG operations (following TGeo)

Status @ ROOT-05 / CHEP-06 – II.

Negative (but not an issue then):

- Over-optimized for geometry rendering
- 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

  Selection, clipping & manipulators tightly knotted.

  Hard to extend (but possible for a price of some ifs), impossible to sub-class or control externally.

Extensions of existing GL viewer

Jan → Aug ‘05: explore GL on central Pb-Pb events

60k tracks, 10M TPC hits ➔ too much data!

interactivity is the key

Early ‘06: first prototype of ALICE display using

ROOT GUI and OpenGL

Apr ‘06: direct OpenGL rendering for ROOT classes

Aug ‘06: two-level selection (container contents)

+ some other minor changes:

decoupling of viewer GUI to follow GED convention

beahaviour of camera during updates, handling of small objs

Direct OpenGL rendering – I.

Manually implement class for GL rendering, eg:

1. For class `PointSet3D` implement:

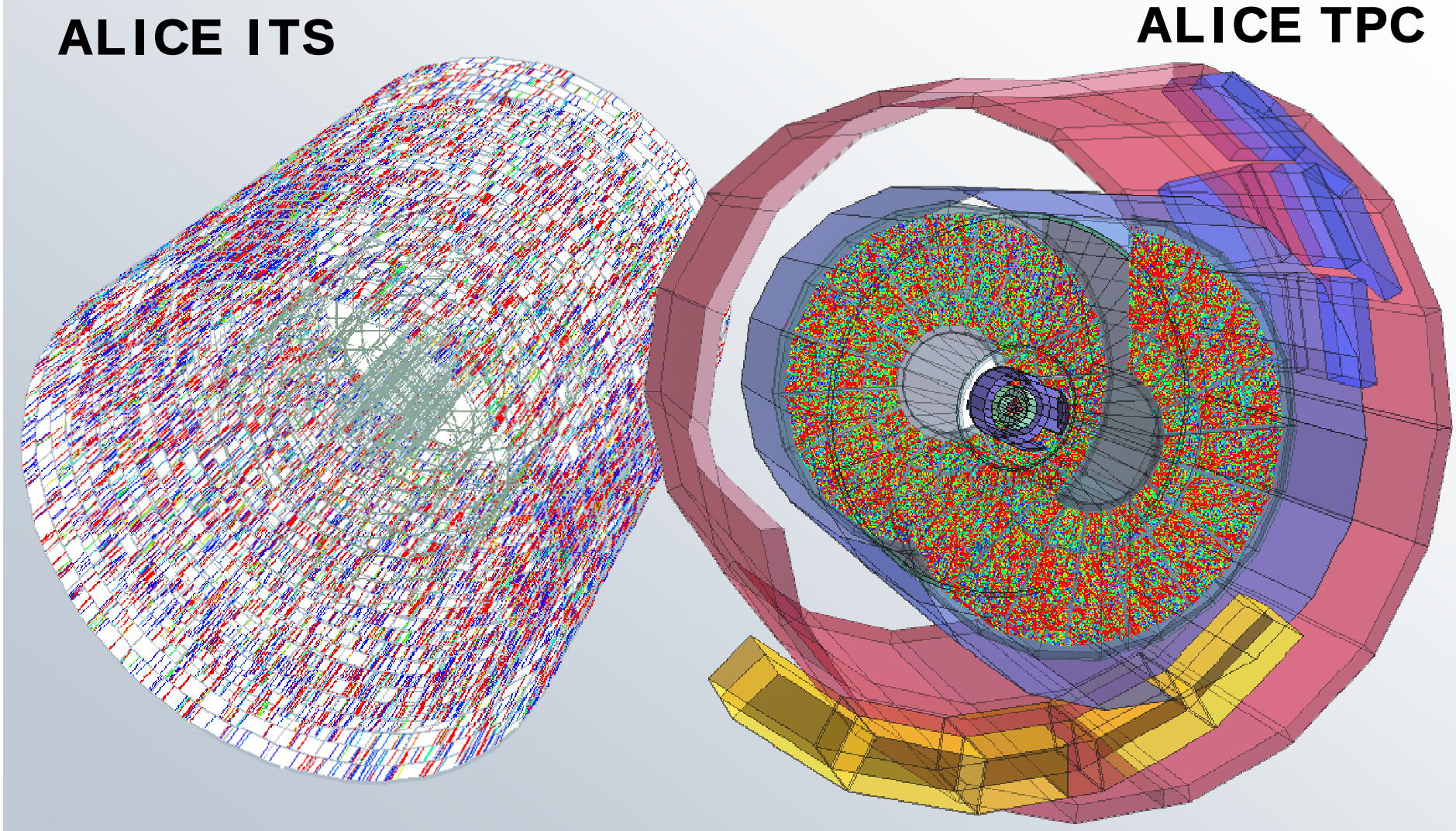
```
class PointSet3DGL : public TGLObject
{
  virtual Bool_t SetModel(TObject* obj);
  virtual void DirectDraw(TGLDrawFlags& flags);
};
```

2. In `SetModel()` check if `obj` is of the right class and store it somewhere (data-member in `TObjectGL`)
3. `DirectDraw()` is called by viewer during draw-pass

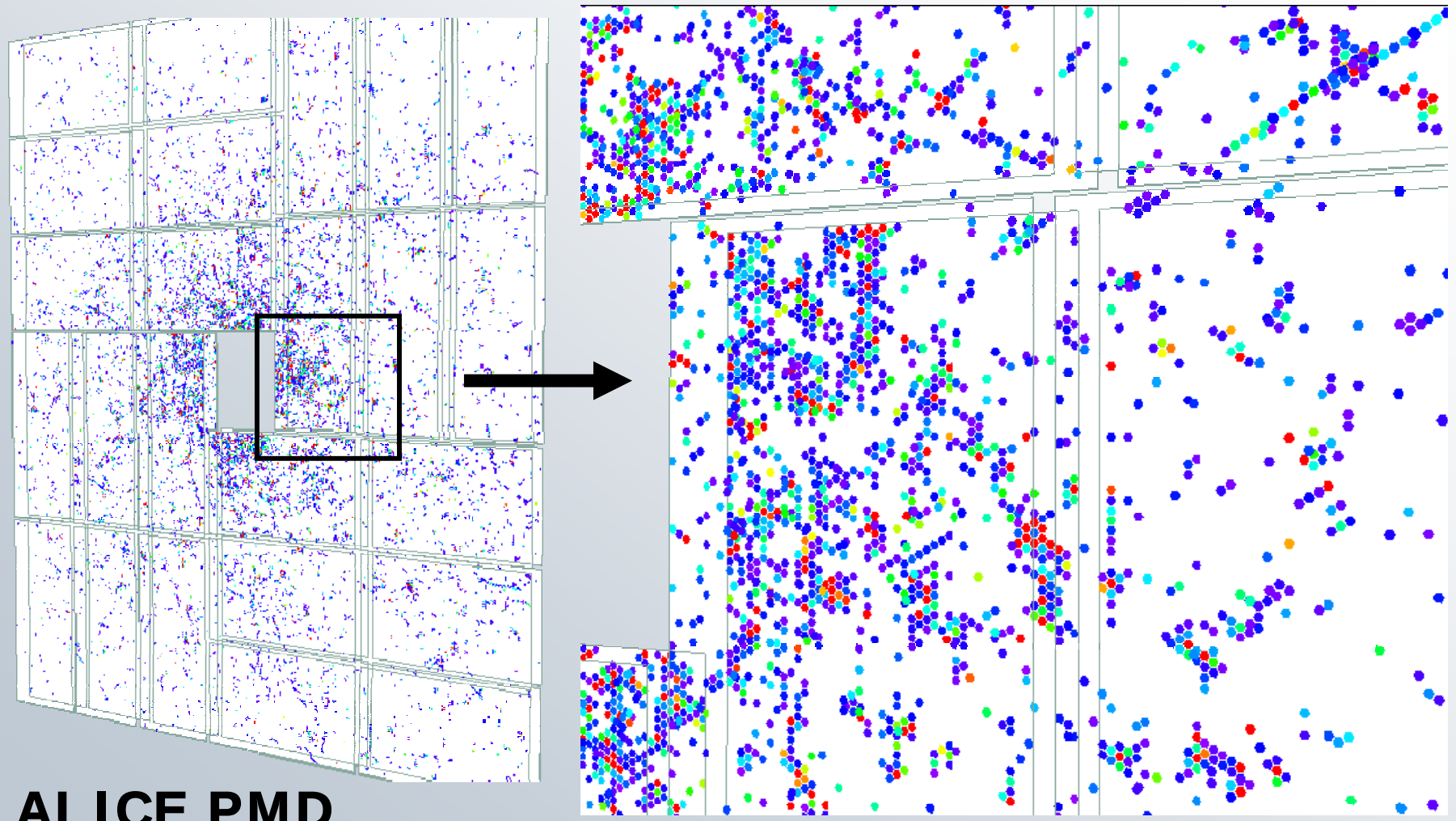
   Here do direct GL calls, change state, draw whatever.

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

Direct OpenGL rendering – II.



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.
3. If found, an object is instantiated via `TClass::New()`
   `DirectDraw()` is called for rendering.
   The GL object can access data of its creator!
4. If not found, negotiation with the viewer continues

Direct OpenGL rendering – V.

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).

But ... this is OpenGL specific solution.

To also support other viewers one could provide:

a) minimal buff-3D representation for each such class
b) similar mechanism for other viewers

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

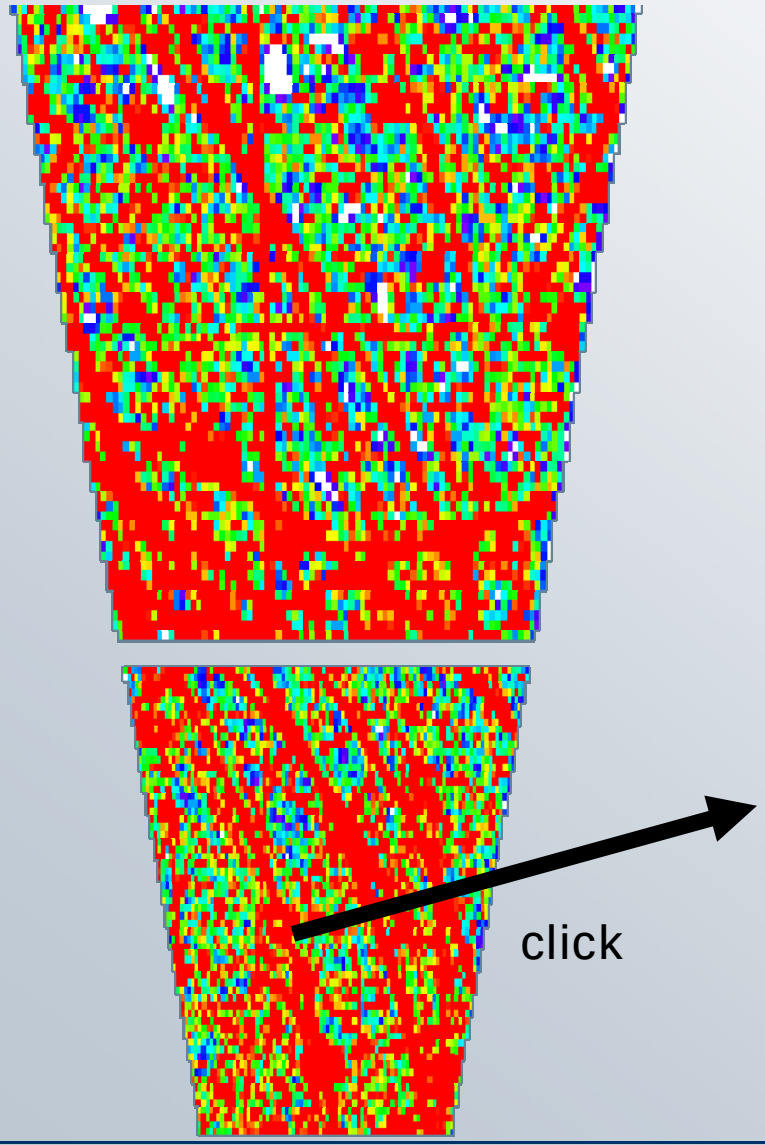
   Investigate, select for further manipulation

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:

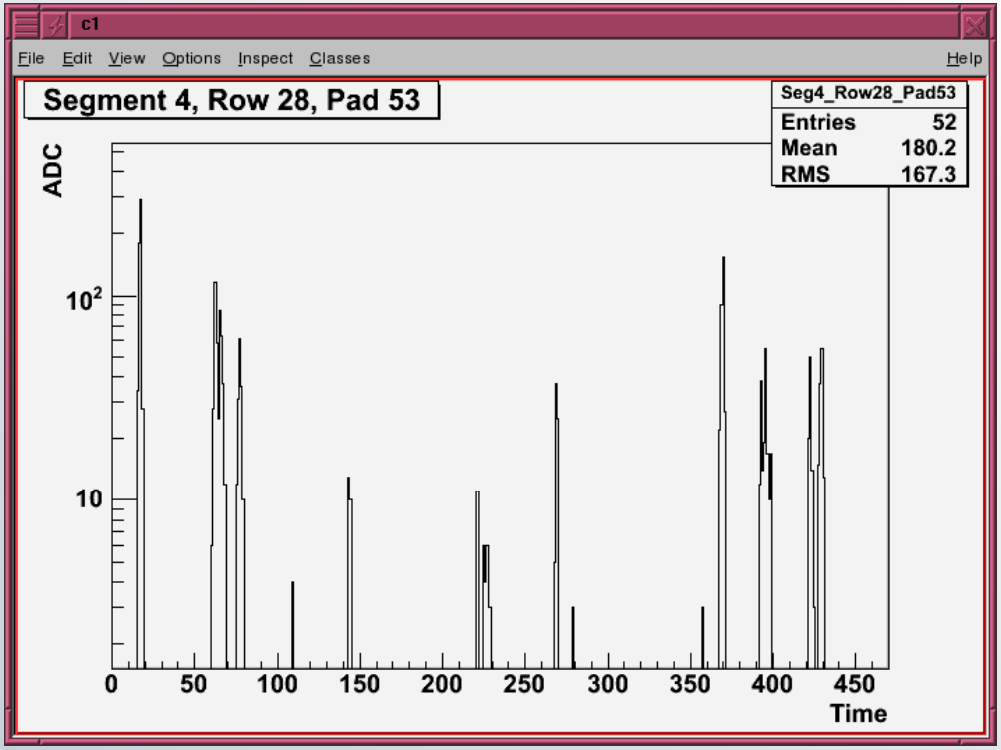
- Optimized version in drawing / first-pass selection
- Special render-path during second-pass (single object!)

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 GL-object-rnr:

```
class TPointSet3D : public TGLObject
{
  virtual Bool_t SupportsSecondarySelect();
  virtual void ProcessSelection(UInt_t* ptr, …);
};
```

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

End of extensions of existing GL viewer

All these changes were evolutionary.

Allowed implementation of ALICE event-display

Summer ’06: major restructuring of GL needed to:

1. Support multi-view displays (shared scenes)
2. Optimize update behaviour for dynamic scenes
3. Modularize input handling (mouse, keyboard)
4. Have appl-specific selection, context and tools

Role of OpenGL .vs. other 3D renderers:

GL becomes the main 3D engine in ROOT!

Others retain minimal support / no new development

RootGL – The Next Generation

December: Why go only half the way? We could:

a) Do all pad-graphics in GL / free mix of 2D & 3D

b) Have all GUI rendered via GL ... err ... not yet.

Mini-revolution needed to keep all options!

Manifest, including concerns from previous slide

I. Provide flexible / general OpenGL infrastructure

1. Support existing ROOT use-cases + new ideas above
2. Include external GL code (non-root based) in ROOT viewer
3. Include ROOT scenes in other environments / toolkits

II. Restructure existing code with maximum reuse

1. Existing functionality kept as a minimal specific case
2. New functionality introduced in parallel classes / implementation

RGL-TNG: Basic infrastructure

1. Low-level GL support [Timur]
   - on/off-screen rendering
   - guidance for feature-use depending on arch
   - frequently used services/functions
2. Decouple base GL from GUI

   split libraries: `libRGL`, `libRGLGui`
3. Keep `TVirtualViewer3D` for compatibility

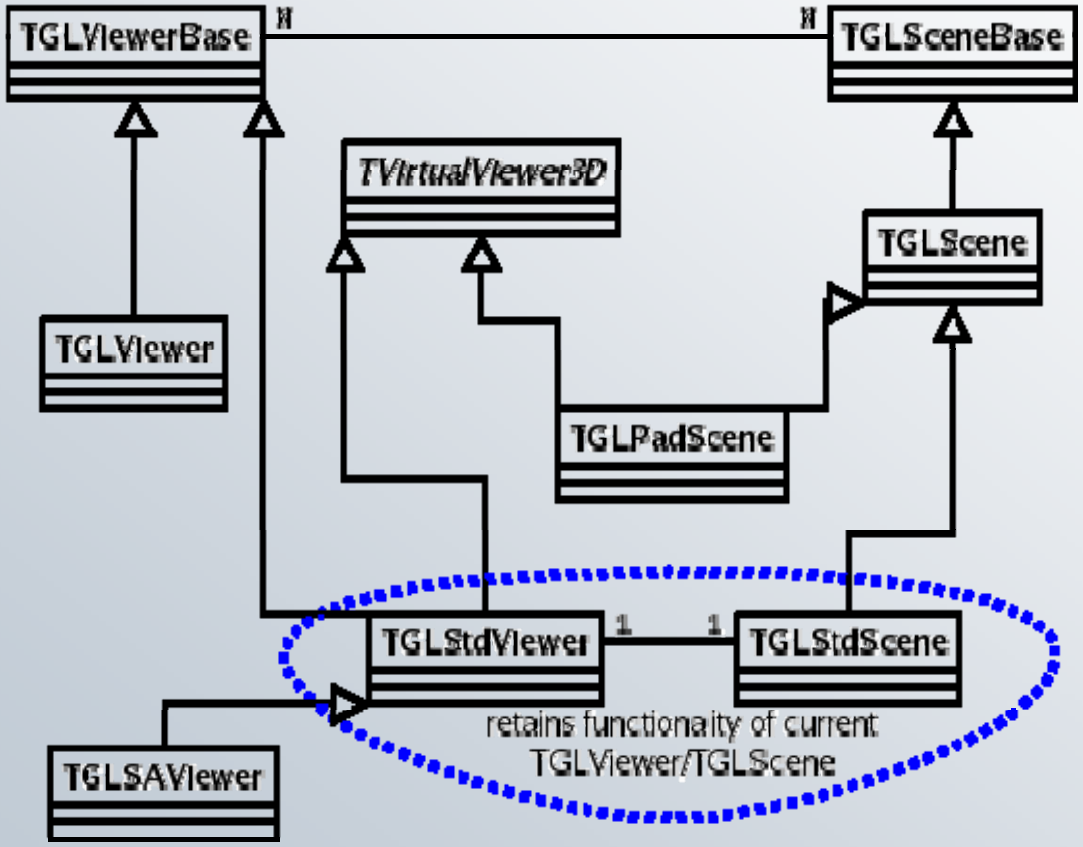
   Default interface for `TObject/TPad::Paint()`
4. Slowly introduce new virtual layers for:

   Passing information on current/selected object

   Event handling / user-interaction

   Partial updates, refreshes, animations

RGL-TNG: 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 logicals/physicals

Cleaned version of current scene

Use this to 'export' a ROOT scene

- **TGLStdScene**

Current TGLScene

- **TGLPadScene**

Natural inclusion of pad-contents:

thus we can service old classes!

Note virtual-viewer3D inheritance

- **TGLViewerBase**: minimal; becomes a collection of scenes + render steering + camera
- **TGLViewer**: add selection & GUI interface (already ROOT specific!)
- **TGLStdViewer**: current TGLViewer, sub-classes from virtual-viewer3D

RGL-TNG: What is already done

- Rewrite of low-level GL interface [Timur]
  - GL-context management
  - Important for sharing of data among several viewers
- Clean-up of logical/physical-shape classes
  - Improved display-list & LOD management
- Clean-up of rendering paths/states
  - Let workers know render-pass details & camera-info
- ~½ of scene code scavenged into base-classes
  - ~¼ is beyond salvation ➔ must re-implement
- Fine-grained per-object updates/adds/removals
  - Missing virtual interface ➔ use TGLScene directly.
  - But these features are GL-only anyway ...

RGL-TNG: Still to be done

For next dev-release (end of April):

- Extract selection code into base-classes
- Provide basic implementation of new viewer with multiple scenes

For the next pro-release (end of June):

- General event-handling / selection mechanism
- Different options for mixing 2D/3D pad graphics

  Use new font library when available [Olivier]
- Some optimizations of rendering on all levels

  LOD calculation

  store scene-draw state for next pass / selection

Conclusion

OpenGL support in good shape & improving

- Last year's development driven by needs of ALICE visualization
  - That's good ➔ heavy-ion events are BIG
  - Interactivity & flexibility
- This year started with a bloodless revolution ... which we hope to mostly end by July.
  - Modularization, better control on all levels
  - Overhead-free scene updates

Good time for further requirements ... let us know!