PROCEEDINGS OF SCIENCE

Object Model to Construct the Mixed "Open Inventor™" / ROOT 3D Scenes

Valeri Fine[1]
Brookhaven National Laboratory
PO Box 5000, Upton, NY 11973, USA
E-mail: fine@bnl.gov

Jerome Lauret
Brookhaven National Laboratory
PO Box 5000, Upton, NY 11973, USA
E-mail: jlauret@bnl.gov

This paper presents an OO model and its implementation, allowing for the creation of OO definitions of complex 3D scenes based on Open Inventor™[1] and ROOT 3D object models simultaneously. Such approach allows us to create the unified visualization software layer to fulfil the different, sometimes contradicting, requirements for Detector Simulation, Event reconstruction and Online Monitoring that could not be otherwise satisfied with existing packages.

The object built upon this model can be saved and retrieved using either ROOT (ROOT macro, ROOT files) or Open Inventor™ ("iv" or "vrml" format files) I/O. They can be rendered with the built-in ROOT TVirtualViewer3D plug-in and with the OpenInventor™-based implementation. To reveal the OpenInventor™ components, a dedicated plug-in was developed. The plug-in and working examples are available as a part of QtRoot project and available for download from http://root.bnl.gov

[1] Speaker

© Copyright owned by the author(s) under the terms of the Creative Commons Attribution-NonCommercial-ShareAlike Licence.

PoS(ACAT)023

PROCEEDINGS OF SCIENCE

XI International Workshop on Advanced Computing and Analysis Techniques in Physics Research
Amsterdam, the Netherlands
23-27 April, 2007

PoS(ACAT)023

© Copyright owned by the author(s) under the terms of the Creative Commons Attribution-NonCommercial-ShareAlike Licence.

1. Introduction

Modern accelerators like the Relativistic Heavy Ion Collider (RHIC) at the Brookhaven National Laboratory and the coming Large Hadrons Collider (LHC) at CERN enable physicists to study the fundamental constituents of matter more closely than ever before. They accelerate beams of gold nuclei or protons to nearly the speed of light before smashing them together to create hundreds of new particles.

Because of the sheer size, and complexity of the new detectors, as well as the huge amounts of data the modern detector is expected to produce, 3D visualization is becoming a major component of any High Energy Physics Frameworks. However it is still not the first concern of the HENP frameworks. We worry whether we will be able to collect, preserve, and re-distribute our hard-earned PBytes.

Nevertheless the final stages of the job are mainly interactive, involving the graphical data representation. The very first steps of the data-taking in the "control rooms" are interactive therefore requiring fast and rich interactive graphical tools.

On the other hand the 3D interactive software is still expansive to design and implement. The man-power constrains have lead us to an introduction of the OO model and its implementation, allowing the creation of OO definitions of complex 3D scenes based on Open Inventor™ and ROOT 3D object models simultaneously. Such approach allowed us to create the unified visualization layer to fulfill the different, sometimes contradicting, requirements for Detector Simulation (Simulation), Event Reconstruction (Reconstruction) and Online Monitoring (Monitoring) that could not be otherwise satisfied with existing packages.

2. 3D visual attributes: one size fits all?

Even though the various visualization tasks in HENP look very similar, from the implementation stand point one still should distinguish:

Task:	**Requirement:**
Online monitoring	Real time
Detector simulation	• Full-fledged geometry (hierarchy) • navigation, • Selection • Comparison of the different versions
Event reconstruction	Event (flat or simple hierarchy) Navigation and Selection on the top of the simplified detector geometry

For example, the attributes "*transparency"* for one and the same object (detector component) can be seen as transparent for the simulation, solid for the reconstruction and invisible for the monitoring

PoS(ACAT)023

Another example of the interesting question to address is whether the attribute *"invisible"* means "*100 % transparent*"?"

The *"Shape"* attributes for the object describing, for example, a calorimeter tower, can be hardwired with the detector (GEANT) description for the "Simulation"; and it can be flexible for "Monitoring" where the shape attributes like size and colour are required to reflect the energy deposit.

Those and the other similar questions are experiment specific. Neither ROOT, nor OpenGL, nor any other generic software tool can address it alone.

2.1 Abstraction layers

The standard approach to satisfy the different requirements above alike is to design and implement several abstraction layers. Fortunately, this task is a relatively simple one for the ROOT-based frameworks [3]. As result the vast portion of our application needs can be met with the quite tiny software layer providing the Coin3D-based (Coin3D is an "Open Source" implementation of the "Open Inventor™" API) plug-in of the ROOT TVirtualViewer3d interface.

The designed plug-in (

Figure 1) was powerful enough to make us free to think about our own applications rather then about the technical details of the 3D object rendering. Since both software components namely STAR framework as well as the Coin3D package use the Qt as its interactive GUI library it was trivial to create the sophisticated interactive application with the advanced 3D rendering engine.

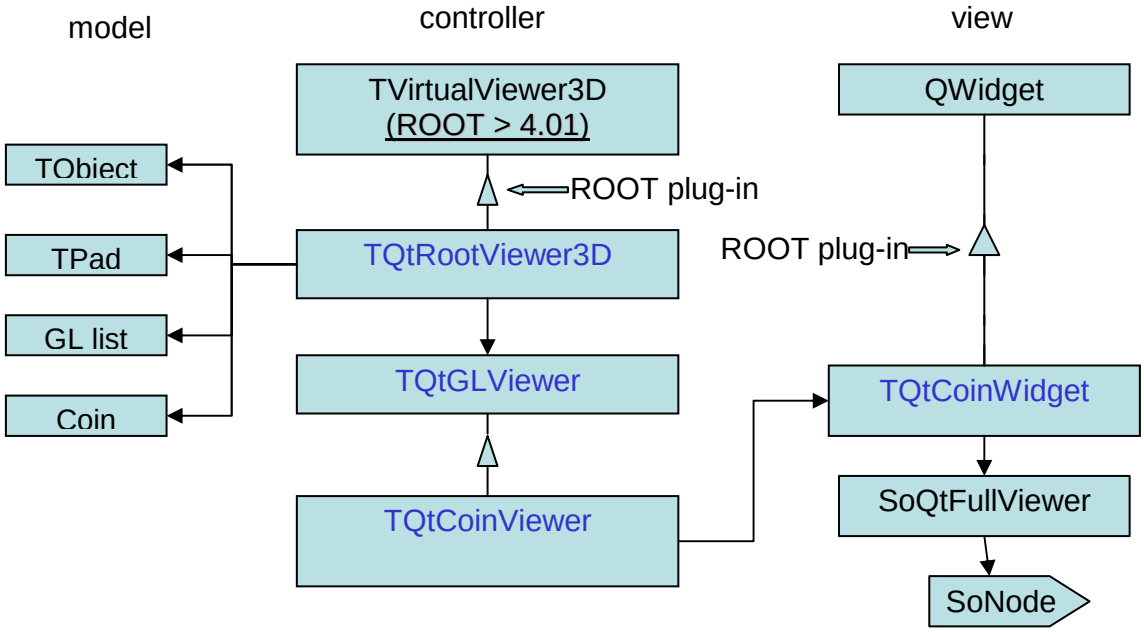


Figure 1. OO Model of the ROOT 3D visualization layer

PoS(ACAT)023

The model is built around of the TQtGLViewer class. This base class emits a few Qt signals (Figure 2.) connected with internal default slots to provide the elaborated 3D object interactive selection (Figure 3, Figure 4).

```
signals:
void ObjectSelected(TObject *, const QPoint&);
void HandleSelected(ULong_t, const QPoint&);
void ImageSaved(QString &fileName, . . . );
```

Figure 2. List of the Qt signals emitted by the TQtGLViewer interface

If needs arise the end-user can connect its own custom slots to the signals and this way enhance the built-in functionality the way that best fits his/her application needs (Figure 5, Figure 10)

To select object one has to

- Turn the Coin viewer "selection" tool on
- Use the left mouse button to point the image on the 3D view

The browser should:

- Highlight the selected shape
- Popup the label with the text provided by the selected object TObject::Info method
- Find the object in the left tree pane of the Geometry Browser and select the found object there
- The object selected in the "tree" list should be painted in the right upper TCanvas widget

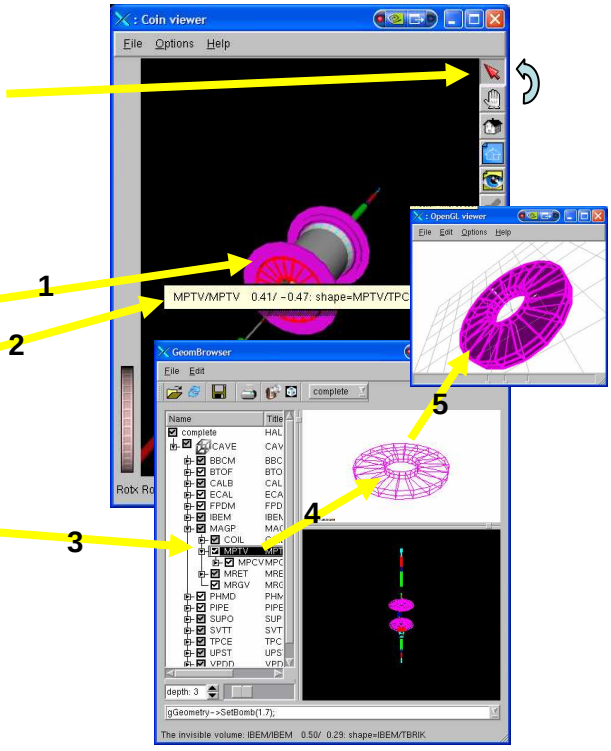


Figure 3. The Default Selection Slot Response

PoS(ACAT)023

The Browser provides another "selection" slot.

The SLOT is activated via the drop-down **"Options"** menu**.**

To select the object, the user should follow the previous slide. The only difference, instead of the label the browser will pop the ROOT **"Context menu"**

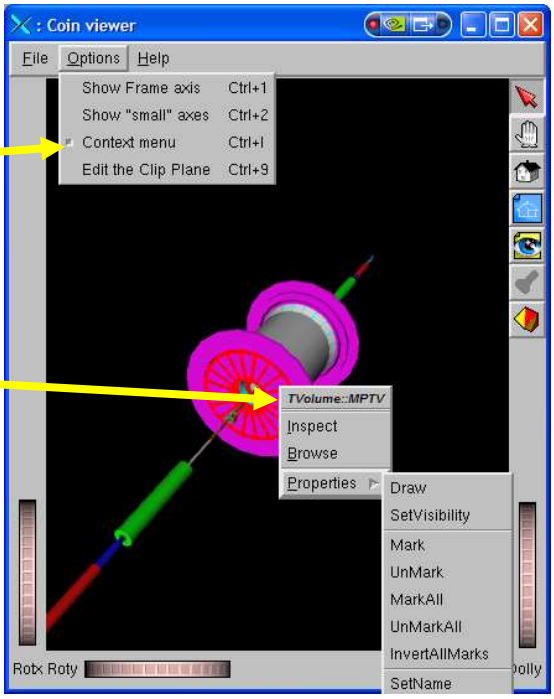


Figure 4. The "Menu" Selection Slot Response

- The SLOT is activated via the icon**.**
- To select the object, the user should follow the previous slides. The only difference, **in addition** to the built-in action, the application is to popup the text edit window with the source code of the selected volume highlighted.

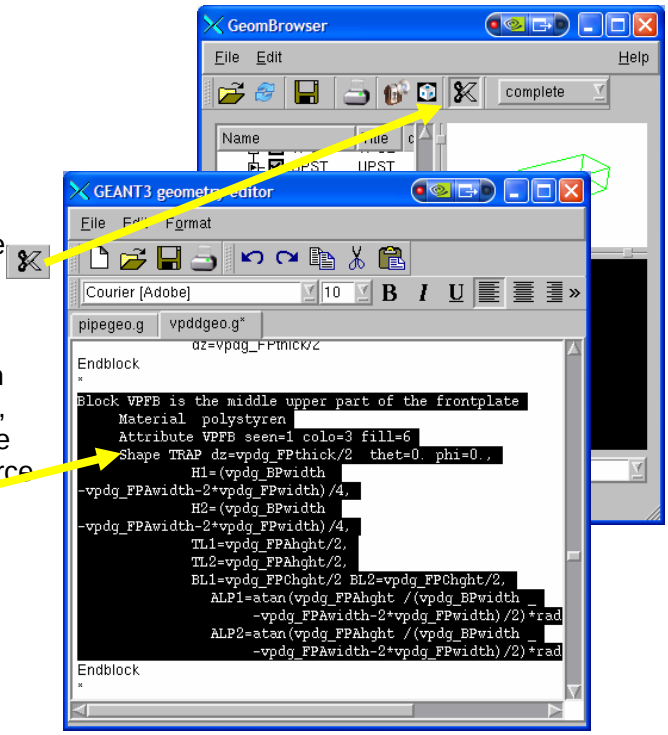


Figure 5. The Custom Selection Response

The Qt-based ROOT 3D plug-ins do allow creating as many OpenGL widgets as your local video hardware can sustain and set the video attributes for each widget separately as the short animation[1] and picture (Figure 6.) demonstrate.

[1] http://www.star.bnl.gov/STAR/comp/vis/animation/GeomBrowser3FilesDemo.wmv

PoS(ACAT)023

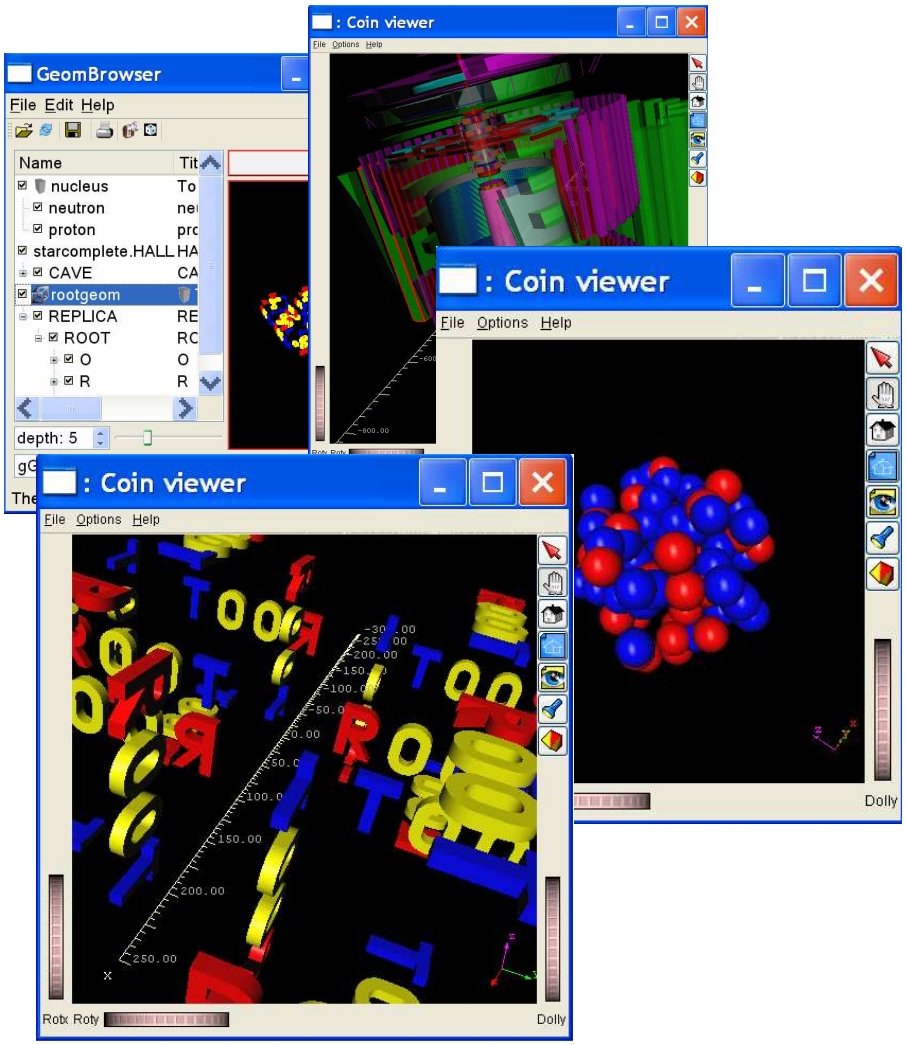


Figure 6. Multiply 3D widgets / attributes

2.2 Coin3D implementation API

The implementation provides not only the advanced GUI interface. To create the custom application one can use the multilevel API also.

- **ROOT API.** The browser uses the standard ROOT plug-in mechanism. This means to use it from within ROOT env to render the ROOT 3D class objects no special API is required. The **`TObject::Draw`** method will be served with the proper plug-in selected via ROOT resource file.
- **Open Inventor API:** To control things programmatically and to render the non-ROOT class objects one can use "Open Inventor™" API [1] (See: http://doc.coin3d.org/Coin/classes.html for details also). The ROOT and non-ROOT objects (including Coin3D animated objects) can be mixed within one 3D scene.
- **Qt API:** Thanks QtRoot and SoQt Coin3D-based widget classes do provide the Qt signal/slot API. This allows using them to create the custom Qt-based GUI, for example

PoS(ACAT)023

for the "Control Room" and to create the animated interactive computer models of the STAR detector useful for students, teaching and public presentations. The widget emits (Figure 3,Figure 4) the Qt signals as soon as the user "picks" some object. The signal provides a C++ pointer to the original ROOT object.

2.3 Input /Output 3D geometry formats

The combination of Qt and Coin3D classes automatically provides the rich set of the input and output data formats:

Table 1. Input / output formats

File format	File name extension
ZEBRA	**`*.fz`**
ROOT macro	**`*.C`**
ROOT binary file	**`*.root`**
Open Inventor™	**`*.iv`**
VRML	**`*.wrl`**

File format	File name extension
All common pixmap formats	**`gif, png, jpg etc`**
Postscript and Encapsulated PostScript	**`*.ps, *.eps`**
VRML	**`*wrl`**
"iv" - "Open Inventor2	**`*.iv`**
Movie	**`*.mpeg`**

2.4 ROOT “Transparency” attributes

ROOT describes the translucent property of its graphical objects via **`TAttFill`** class as follows:

4000	the object is transparent
4001 – 4100	the object is for 100% transparent to 100% opaque.

In addition, the ROOT 3D classes provide an extra “visible” attribute.

There are several ways to see the transparency attributes for ROOT object. For our model we decided to treat:

- Full 100 % transparent object – wired and “unpickable” image
- 99% transparent object – wired and “pickable” image
- > 99% – translucent solid image.
- Invisible – “invisible” means no image at all.

PoS(ACAT)023

This approach helps us to accommodate the various needs to interact with the 3D object without any change of the original ROOT object model (Figure 7, Figure 8).

```
trap->SetLineColor(4);
trap->SetFillStyle(4001);
```

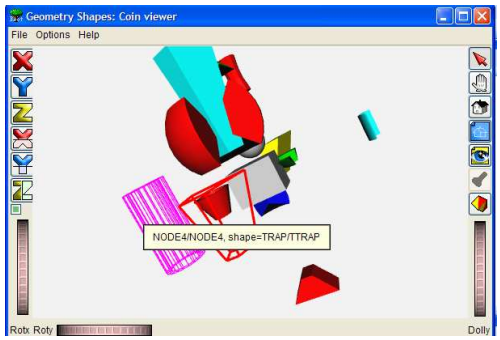


Figure 7. ROOT FillStyle 4001 makes the object wired and availabe for the pick operation.

```
tube->SetLineColor(6);
tube->SetFillStyle(4000);
```

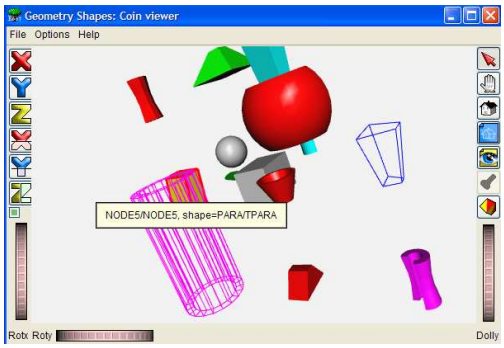
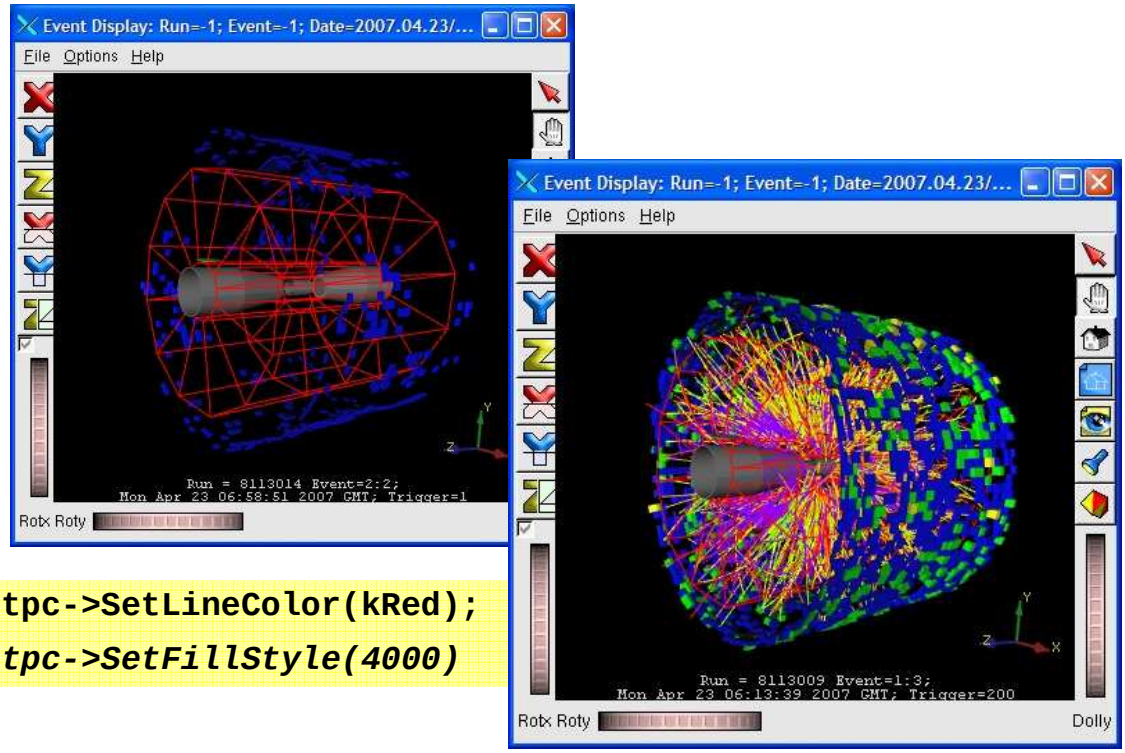


Figure 8. FillStyle 4000 makes the object invisible

The next picture (Figure 9) shows the usage of this attribute for the STAR Online Event Display.

PoS(ACAT)023




```
tpc->SetLineColor(kRed);
tpc->SetFillStyle(4000)
```

Figure 9. STAR Online Monitor using non-clickable 4001 style to outline TPC

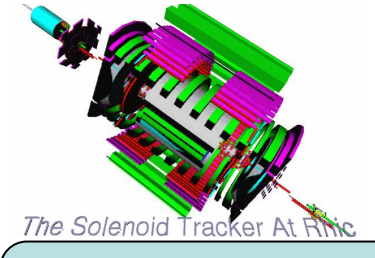
3. Object Model to Construct the Mixed 3D scenes

The introduction of the Coin3D-based ROOT-plugin opened the access to the OpenInvetor 3D manipulation facilites. The ROOT class library lacks of. On the other hands the ROOT portion provides the advanced OO I/O system to save and restore the combined ROOT/ Coin3d complex objects. The model is designed the way that one

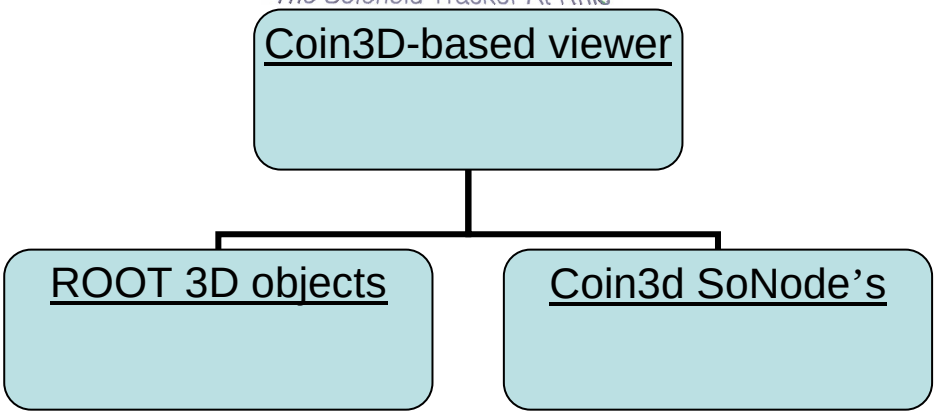
- Needs change neither ROOT 3D object model, nor the 3D virtual viewer model but the "attributes visual implementation" only
- needs no change of ROOT object model, to mix the ROOT and Coin objects at the rendering time
- can enhance the ROOT object model to provide the new attributes to be treated by the dedicated viewer (but preserve the backward / forward compatibility) (Table 2)

PoS(ACAT)023

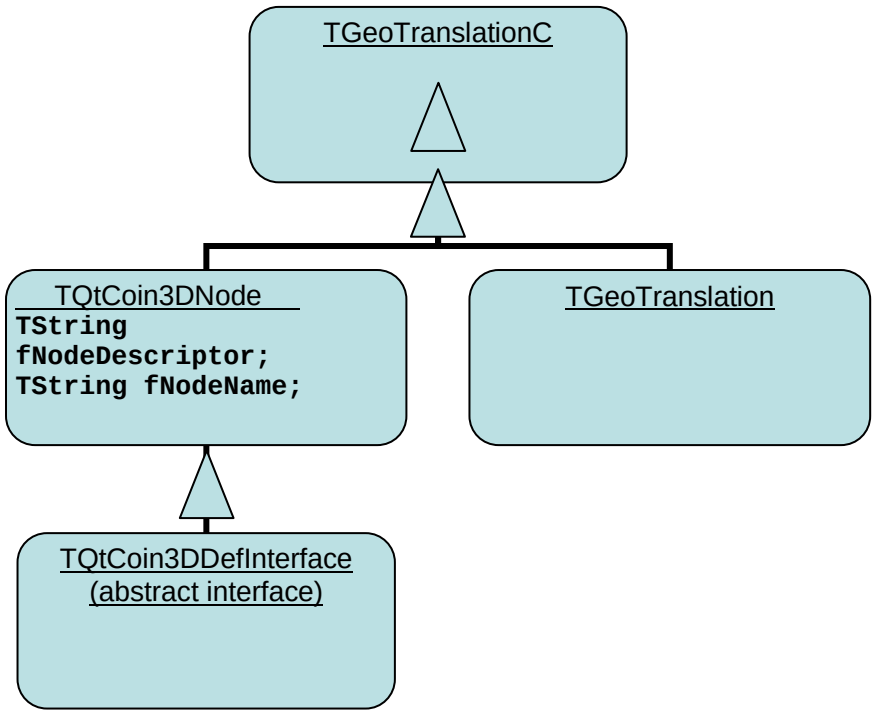
Table 2. Mixing ROOT and OpenIventor™ objects at the rendering time



Mix the ROOT and Coin3D objects at the rendering time within Coin3d viewer



To provide this the simple OO model was introduced and implemanted:



PoS(ACAT)023

The switching from the regular ROOT OO model to the mixed one is as simple as replacing the the class name **`TGeoTranslation with TGeoTranslationC. The figures below show the small corrections to the standard ROOT test rootgeom.C and the OpenInvetor file animating the original ROOT picture:`**

```
TGeoTranslation *replica1=new TGeoTranslation(-150, -150, 0);
```

```
TGeoTranslationC *replica1 =new TGeoTranslationC(-150,-150,0);
replica1->SetFileName("replica1.iv");
```

replica1.iv:

```
#Inventor V2.1 ascii
Rotor {
    rotation 0 1 0  0.1
    speed 0.1
    on TRUE
}
```

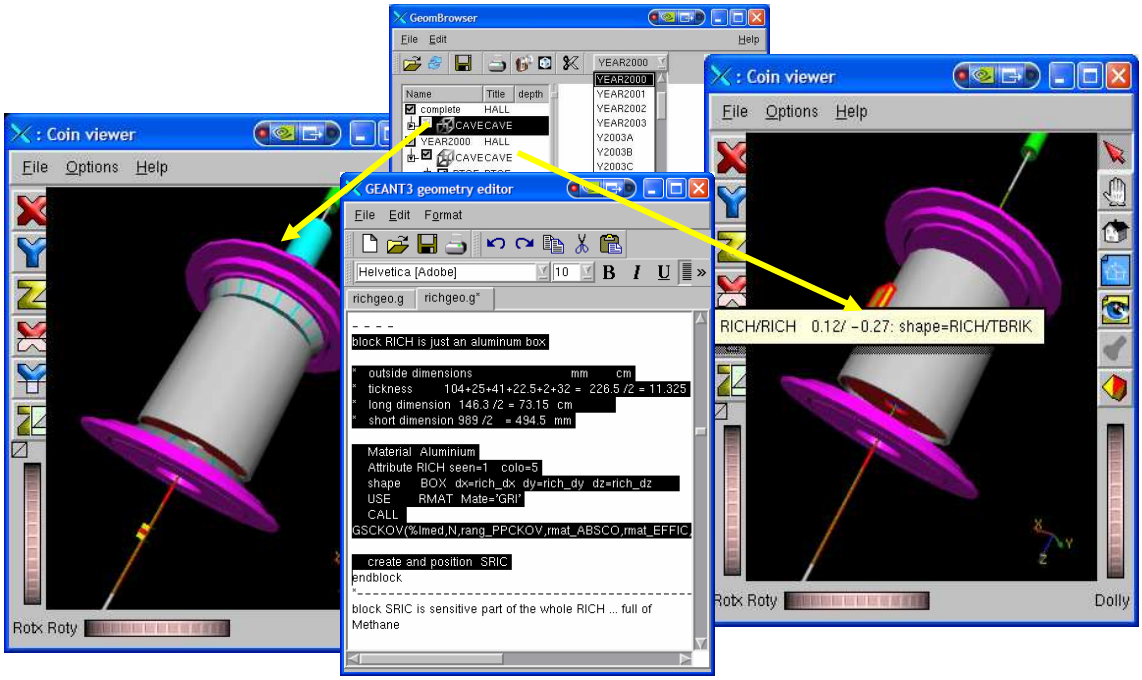


Figure 10. Use 3D graphics to debug the simulation code

PoS(ACAT)023

4. STAR applications

The present OO model and it implementation became the foundation for several STAR applications. The Figure 10 shows the screen shot of the detector geometry editor that allows not only getting high quality OpenGL pictures of any piece of the detector but comparing several versions of the GEANT3 geometry layouts side by side.

The online monitor (Figure 11) have been in use for 3 last STAR runs and it was fast enough to visualize the internal detector state in online and feed the STAR Web page with current collision view in near real-time.

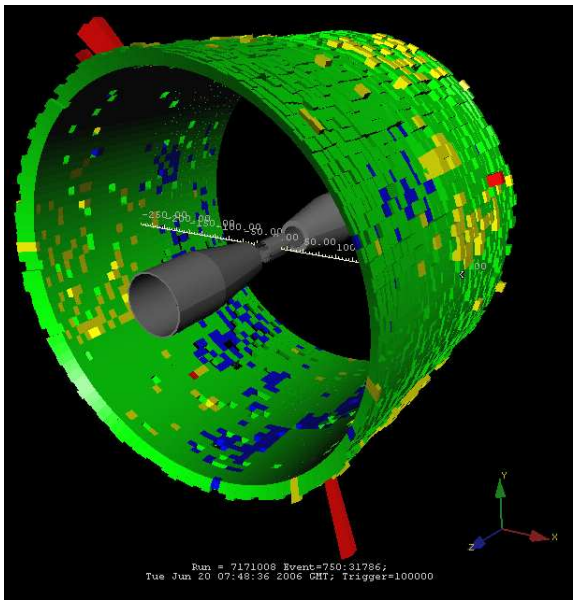


Figure 11. Screenshot of the STAR Online Event Display

STAR offline event display (Figure 12) has become a tool fo choise to debug the complex event recostruction algoritms.

PoS(ACAT)023

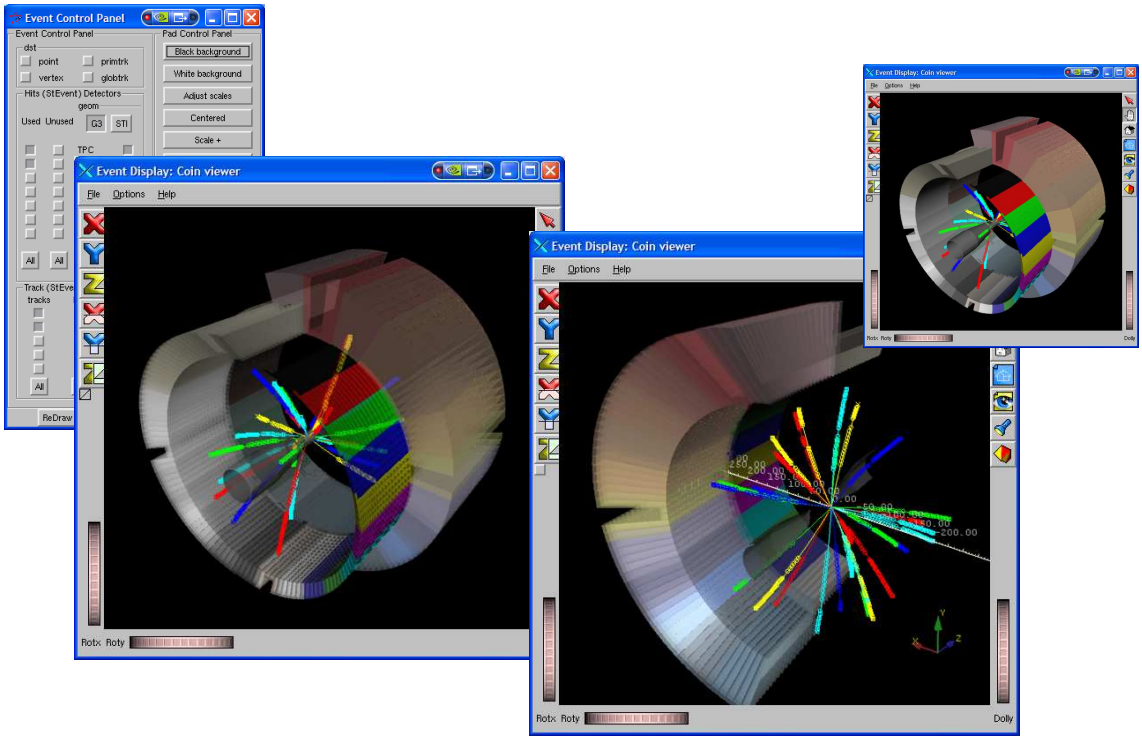


Figure 12. Use the OprnInventor-based viewer for offline event reconstruction

5. Conclusions

- The object built upon this model can be saved and retrieved using either ROOT (ROOT macro, ROOT files) or Open Inventor ("iv" or wrl" format files) I/O. They can be rendered with the built-in ROOT TVirtualViewer3D plug-in and with the Open Inventor-based implementation also. To reveal the "Open Inventor" components, a dedicated plug-in was developed.
- The model and its implementation do allow STAR to effectively combine the ROOT OO detector description and Coin3D sophisticated 3D scene visualization features.
- The plug-in is a part of the QtRoot project and it is available to download from QtRoot project Web size and CVS repository (see: http://root.bnl.gov) (Windows binary do include Coin3D plug-in)

References

[1] Josie Wernecke, *The Inventor Mentor: Programming Object-Oriented 3D Graphics with Open Inventor™, Release 2*, Addison-Wesley Publishing Company, ISBN 0-201-62495-8

[2] S. Siemen, *Interactive 3D Worlds with Coin*, in http://www.linux-magazine.com/issue/28/Coin3D_Part1.pdf, March, 2003

[3] V. Fine, *STAR Framework and Visualization*, in proceedings of *ROOT 2000 International Workshop*, CERN, Geneva, February, 2000

[4] P. Nevski, *STAR Simulation*, in proceedings of *ROOT 2000 International Workshop*, CERN, Geneva, February, 2000

PoS(ACAT)023

[5] R. Brun et all, *ROOT OO model to render multi-level 3-D geometric objects via an OpenGL*, in proceedings of *the VII International Workshop on Advanced Computing and Analysis Techniques in Physics Research*, Chicago, October 16-20, 2000

[6] V. Fine, *Visualization of the ROOT 3D class objects with Open Inventor-like viewers*, in proceedings of *IX International Workshop on Advanced Computing and Analysis Techniques in Physics Research*, December 1-5, 2003

PoS(ACAT)023