CDF – Rome Group – Internal note – October 15, 2004

A "light" display for CDF

D.Carrabino [1], M.Formica [2], P.F.Loverre [3], S.Sarkar [4]

[1] Student – Informatica – Università La Sapienza – Roma – e-mail danilo.carrabino@gmail.com
[2] Student – Informatica – Università La Sapienza – Roma – e-mail: matteo.formica@fastwebnet.it
[3] INFN – Sez. di Roma1 – e-mail: loverre@roma1.infn.it
[4] INFN – Sez. di Roma1 – e-mail: subir.sarkar@cern.ch

Abstract

We describe a preliminary version of a program intended to provide a display of the central region of CDF. The program is based on ROOT. It is intended to run on TopNtuples but can be easily extended to ntuples of other physics groups in CDF. It displays charged particles trajectories, and calorimeter energies.

1 Introduction

The powerful CDF event display programs runs on full reconstructed events which make it a little inconvenient for certain types of analysis. Our goal is to provide a short program for event display, which uses reduced info (reconstructed tracks, and calibrated tower information) and can therefore run directly on reduced data sample (e.g. short n-tuples). The display program is intended to be a simple tool for physics analysis, mainly of central jet data. The program that we present in this paper is not yet finalised, but we think that the present version can be used as a starting point for improvement. We present in section 2 an overview of the program. In section 3 we describe in more detail the features of the program and we present a kind of user's manual of the display. Section 4 gives the location of the code and a description of the options for inputting the data. Conclusions and a summary of the work needed to finalise the program, are presented in section 5. The appendix (section 6) describes with the code in detail (methods and algorithms).

2 Overview

Our event-display program is written in C++ and is based on the tools made available by ROOT. The program produces two different representations of the events, as shown in figures 2.1 and 2.2. We shall refer to the display in figure 2.1 as to the R-phi representation, and to the eta-phi representation for figure 2.2. The eta-phi display consists simply in a planar grid of the eta and phi variables, where the energies measured in the calorimeter are displayed with towers of different colors for the hadronic and electromagnetic parts. The user can operate on this diplay, by zooming and performing rotations and displacements. The R-phi representation is more complex. The user can switch between a two-dimensional view of the event, and a representation in space. The two-dimensional representation is a fix display of tracks and towers, projected in the X-Y plane. The three-dimensional representation (fig.2.3) displays a cylindrical grid, with radius corresponding to that of the tracking region of CDF. Particles trajectories are displayed inside the cylinder. Calorimetric towers are positioned on the grid.

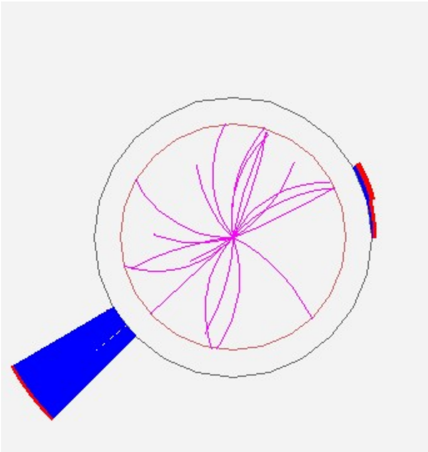


Figure 2.1

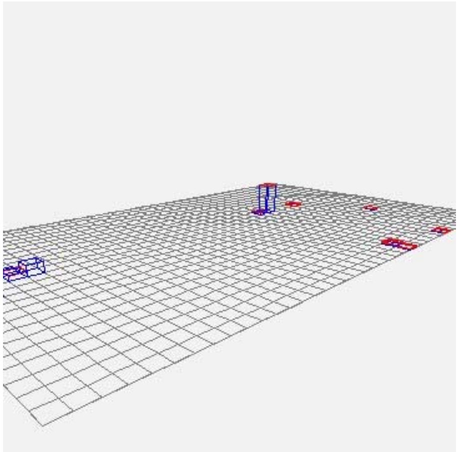


Figure 2.2

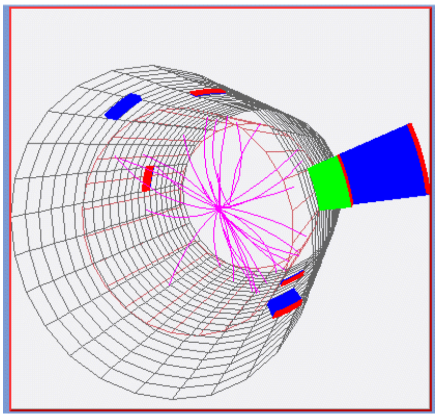


Figure 2.3

3 - Layout of the display – User’s manual

The graphic layout is based on the Root Graphic User Interface **(GUI)** widgets. The display program can be run locally on any desktop where ROOT is installed. The display is activated by a call to ./viewer . Input to the display are information on reconstructed tracks from the interaction vertex, and towers energies (em, had). The input format can be of different types and will be discussed in section 4. By running the program, the user will get on the screen a display of the kind shown in figure 3.1 . The main window (an object of the CDF_Viewer class, which extends the TGMainFrame class) has been designed to contain all the GUI widgets to be used: the menu-bar, an Embedded Canvas, two editable text areas, the button panel and the status bar.

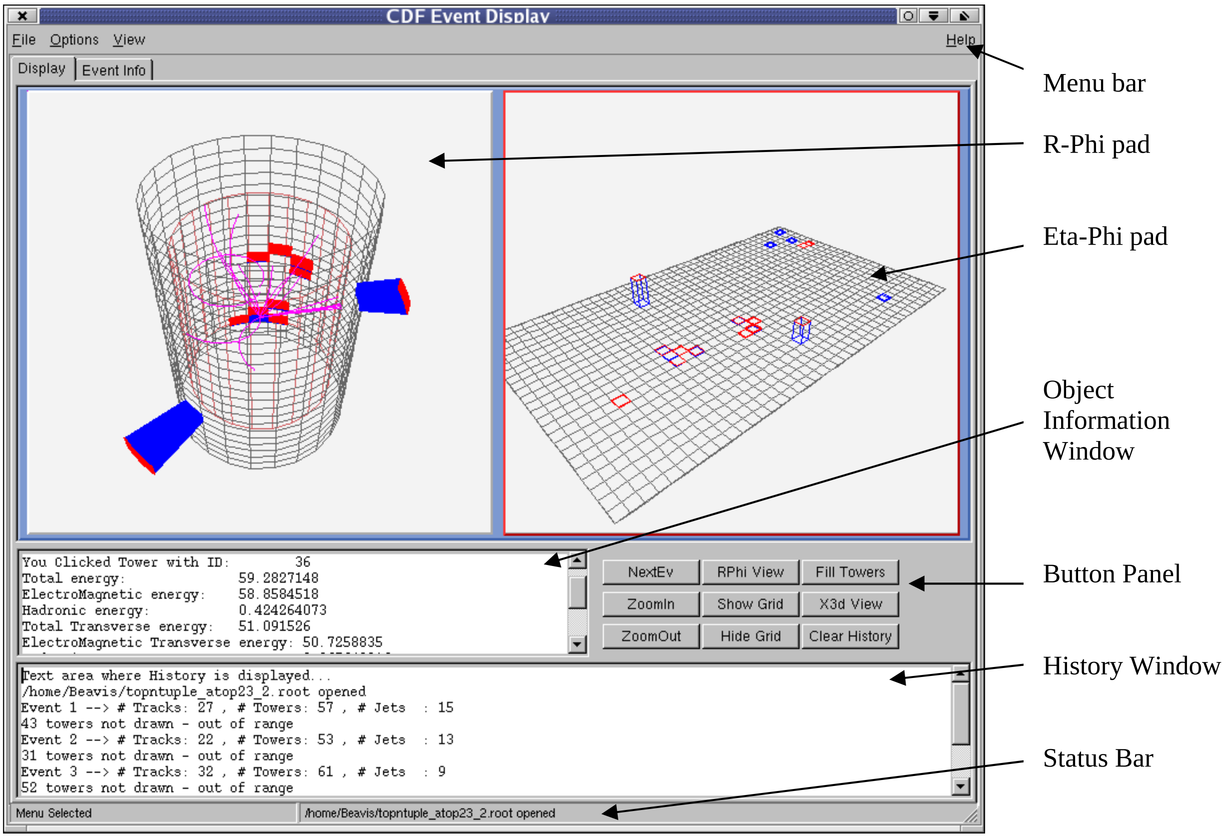


Fig.3.1

The details of the elements shown in figure 3.1 are the following.

a - Menu-bar

The bar allows for three sub-menus

- Sub-menu “File” → addresses New, Save Canvas, Print or Quit ROOT
- Sub-menu “Options” with ROOT Object Browser
- Sub-menu “View” → contains the items corresponding to those indicated in the Button Panel

b - Display tab (object of the DisplayTab class)

This section is made of a TCanvas including the two display pads, *RPhi_pad* and *Grid_pad*, together with all the other features of the display. The user can interact with the display. The interactive options are the following.

- Click on the displayed objects (tracks or towers) to get numeric informations on those objects.
- Zoom in and out.
- Rotate the full display in space.

Note that the RPhi_pad allows to switch between the two different representations (cylindrical grid <-> r-phi projection) described in the previous section.

The other parts of the Display tab are the following.

b.1 - Object Information Window (oggetto TGTextEdit)

This is an editable text area where the informations on the selected object (track or tower) are shown. This area is reset by any new ‘click’ on an object.

b.2 - History Window (TGTextEdit object)

This editable text area contains the informations on the current event (evt nb, nb of tracks, of towers, etc.), as well as informations on the current operations (opening of files, menu, activated buttons). This area can only be reset manually, by pushing the CLEAR button.

b.3 - Button Panel (TGButtonFrame object)

The button panel is shown again in figure 3.2. The buttons (TGTextButton) have the following functions.

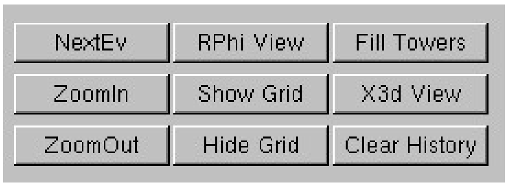


Figure 3.2: The Button Frame

NextEv: Read a new event from input, and display it
Rphi View: Switch between the two representations in the R-phi pad (R-phi <-> cylinder)
Fill Towers: Enable/disable color filling of towers in the left pad
ZoomIn/Out: Zoom In or Zoom Out on the selected pad
Show/Hide Grid: Enable/disable grid drawing on both pads
X3d View: Shows the selected pad with X3D Viewer
Clear History: Erase History Window content

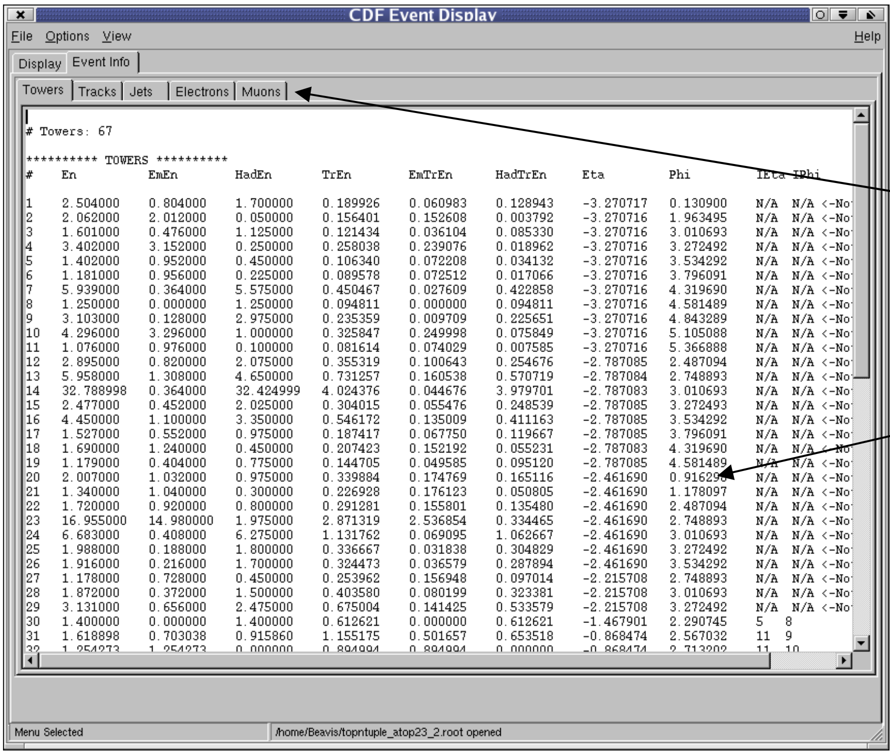
The X3D Viewer is a special tool of ROOT which allows to see another grafic representation of the currently selected pad and will be described later.

c - Status Bar (TGStatusBar object)

The Status Bar shows, on the left, the last user operation on the interface and, on the right, the last open file.

d - Event Info tab (object of InfoTab class) (figure 3.3)

The Event Info tab has five sections ("sub-tabs") which allow to display (within TGTextEdit) information on Tower, Tracks, Jets, Electrons and Muons, directlty from the input file.



The five "sub-tabs" each containing a text area with information on towers, tracks, jets, electrons and muons

Data shown in the editable text area

Figure 3.3 : Event Info Tab (Displayed sub-tab: "Towers')

The display pads extend to the following regions.
- R-phi pad. Inner radius = 138 cm, Outer radius = 170 cm,
- inner cylinder axis 87 cm < z < 397 cm, outer cylinder axis 0 < z < 484 cm,
- eta-phi pad. Grid: $-2 < \eta < +2$ Steps: $\Delta\eta = 0.1$, $\Delta\phi = 15°$

For the currently selected pad, the X3D-VIEW button makes available a different graphical representation of the pad itself; X3D are precompiled graphical libraries already present in ROOT (like OpenGL libraries), and our display can interface with them, through the “X3D VIEW” button. Furthermore, in X3D it is no more possible to select tracks or towers, but rotating and zooming are still available. Examples of the X3D-VIEW display are shown in figure 3.4.

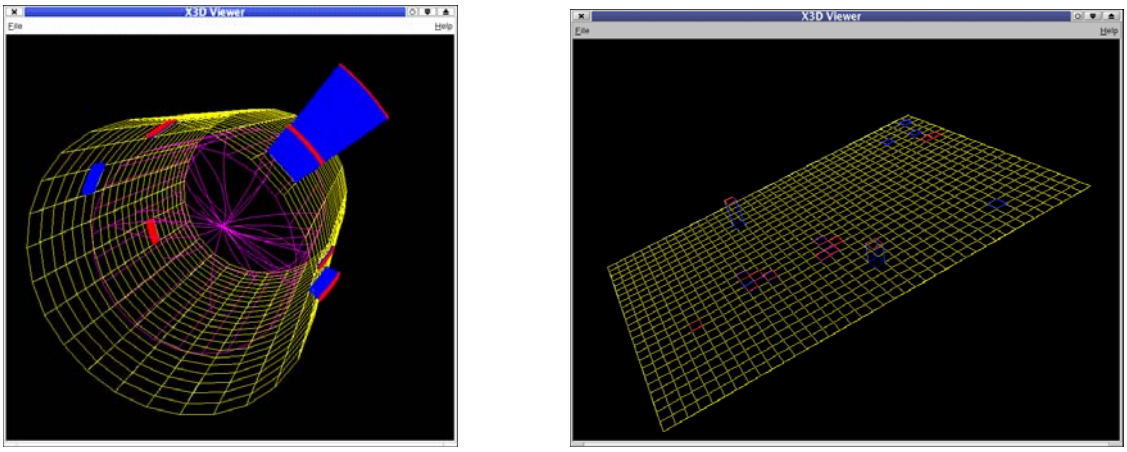


Figure 3.4 : R-Phi and Eta-Phi visualised with the X3d Viewer

4 – Running the program – code repository and input format

The code is available under **`/afs/infn.it/roma1/user/formicam/public/final_viewer .`**
Before running the Light Display locally, the user has to make sure that ROOT is installed in his Linux system; then he can download from the already mentioned afs area the source code (together with the Makefile) in a destination folder, compile them and execute (by typing ./viewer). Instead, to run the display remotely, it’s enough to open a ssh connection to a machine of the domain roma1.infn.it, to enter the previously mentioned directory, and to type ./viewer from the line command of a Linux system.

For each event, the displayed quantities are data about tracks, towers and jets. The display program looks for these data in the following matrices

```
Float_t EnergiaEm[G_EtaRange][G_PhiRange]
Float_t EnergiaHad[G_EtaRange][G_PhiRange]
Double_t trackInfos[G_EtaRange*PhiRange][8];
Double_t* towerInfos[G_EtaRange][G_PhiRange];
Double_t jetInfos[100][8];
```

For each `i` and `j`, the matrices `EnergiaEm[i][j]` and `EnergiaHad[i][j]` have to be filled, respectively, with the values of the Electromagnetic and Hadronic energy in the cell `[i,j]` of the grid.
The `trackInfos` array have to be filled with the informations necessary to represent the tracks (P, Pt, Eta, Phi, curvature, etc.); the `towerInfos` array has to be filled with the values necessary to represent the towers `(E, EEM, Ehad, Et, EtEM, EtHad, Eta, Phi, iEta, iPhi)`.However, for more details see the sections 6.3 and 6.4. As the jets are not represented in the actual version of the display, the values in the array `JetInfos` are just needed to fill the ‘Jets’ section of the EventInfo tab.

In the present version of the program the user has two options for the input. In the method `EventDisplay::MatrixFeeding` data are read from a ROOT file. By using `EventDisplay::MatrixFeeding2` data are read from an ASCII file. In the following we give some detail on the two options.

Reading from ROOT file (ROOT Tree) - The program has been designed to read from Top_ntuples. The reading is handled by the class TopTree.C The operations of file opening and objects forming are performed within the CDFViewer class. Ttree objects are created by the Get() method:

```
Ttree *tree=(TTree*) f.Get(“TopTree”)
```

Then, the TopTree object is formed:

- `TopTree *tt=new TopTree(tree)`

After the initialization of the energy matrices, there is the call to the `EventDisplay::ReadEvent()` method to load all informations in the TopTree.

Reading from ASCII files – The second option for the input format consists in ASCII file. An example of this format is shown in the following, for the case of an event with two tracks, two towers and two jets.

```
== Event # 1
== ntracks =  2
 Trk    P   Pt    Eta    Phi    Curv          D0 Z0    Chi2    Prob       nCot       nSI
  1    50  49.9  0.001  1.0    +0.0000423848  0  0    609     0.000      571610921  50595855
  2    60  59.9  0.001  4.15   -0.0000353088  0  0    1       1          1          1
== nTowers = 2
  Idx      E      EEM     EHad      Et    EtEM     EtHad     Eta      Phi     iEta iPhi
   0       52      52      0        51.9   51.9       0      0.001    1.0     1    1
   1       63      63      0        62.9   62.9       0      0.001    4.15    1    1
== nJets = 2
  Idx   E        P         Et       Pt       Eta      Phi      EMF        ntrks
   0    128.02   127.83    51.71    51.63    1.556    2.926    0.380      5
   1    57.20    57.17     29.66    29.65    1.274    1.254    1.000      3
```

5 - Conclusions

We have described the features and the method of operation of our "light display" program. The program is intended to provide in a simple way a display of particle trajectories and calorimeter energies in the central region of the CDF detector. Though the general structure of the program is fully working, a few aspects of the program have to be revised, and some other features implemented, to make the program a really useful tool for analysis (mainly for the study of jet-physics, as initially planned). The main aspects on which more work is needed are:

- revise the geometry of the "R-phi" display, which in the present version contains some wrong relation between the z, η and θ variables
- revise the representation of towers (scaling and behaviour under rotations with the X3D-View) of the "R-phi" display
- improve the representation of particle trajectories, correctly taking into account initial and final track points
- implement the display of information about reconstructed jets

6 - Appendix – Description of the code

6.1 – Basic classes

We first describe the basic classes used in the program.

- **`CDF_Viewer`** `class: public TGMainFrame`

This class contains the definition of the main body of the GUI. The task of its constructor consist in designing a panel and in mapping all the contained sub-panels: the menu-bar, with *DisplayTab* and *EventInfoTab*, and the status bar. The class includes methods for reference rendering (*getEventInfoTab()* e *getDisplayTab()*), the *ProcessMessage* method to get events generated with the menu bar options, the *ResetCDFDisplay* method, and various methods for handling files.

- **`DisplayTab`** `class: public TGCompositeFrame`

This class handles the event display. The constructor calls the *CreateDisplay* method which, based on the *TGCompositeFrame,* provides the inclusion of a TrootEmbeddedCanvas, a *fAFrame,* and a HistoryWindow.
The method *InitPads()* creates the two display pads (*Rphi_pad, Grid_pad).*

- **`TextTab`** `class : public TGCompositeFrame`

This class is used to form the inner tabs of the EventInfoTab

- **`EventInfoTab`** `class : public TGCompositeFrame`

The EventInfoTab (extending TGCompositeFrame) consists of a frame with 5 sub-tabs: `fTowersTab`, `fTracksTab`, `fJetsTab`, `fElectronsTab` e `fMuonsTab`. The same class includes the methods *getTowersTab(), getTracksTab(), getJetsTab(), getElectronsTab(), getMuonsTab()*.
(all the above classes are located in the *CDF_Viewer.cxx* file together with the **`PrintSetup`** and **`TextViewer`** classes)

- **`EventDisplay`** `class : public TObject`

The EventDisplay class is the core of the program. The class executes many tasks. The most relevant are: drawing of the grids (*CreateGrid* and *TubeDraw* methods), drawing of tracks and towers (*DrawHelix* and *TowerGenerator*), input data reading (*MatrixFeeding* and *MatrixFeeding2*) .

- **`New_Node`** `class : public TNode`

This class has been written to overwrite the *ExecuteEvent* method and make possible the interaction with the objects created by that class (enabling the functionalities MouseEnter, MouseLeave, Button1Down etc.)

- **`CDF_Helix`** `class: public THelix`

The CDF_Helix class has been implemented to redefine the ExecuteEvent method of the THelix class, so to provide the possibility to interact with the tracks by mouse clicking (MouseEnter, MouseLeave, ButtonDown).

- **`TopTree`** `class`

This class is only used to read events from a ROOT file. The class includes the *LoadTree* method. The NextEvt button activates the call to the MatrixFeeding method, which makes the LoadTree of the TopTree object.

6.2 Drawing the grids

After placing the pads on the main canvas (fACanvas in the EventDisplay class), the next task consists in drawing the grids.

The Eta-phi grid – The rectangular Eta-phi grid is created in the `EventDisplay::CreateGrid` method. The grid is made of TPARA objects, rectangular prisms of zero height. The TPARA objects, disposed one next to the other, are contained inside a TNode, and the whole system is inside an invisible TBRIK object. The cells of the grid are linked to the TNode matrix element:
`TPARA *br[G_EtaRange][G_PhiRange]` related to
`TNode *nodo[G_EtaRange][G_PhiRange]`
A first cycle creates the TPARA objects and inserts them in the *br* matrix. A second step is needed to create the nodes containing the TPARA objects. The *name* of the object shape (contained in the auxiliary strings *aux* and *aux2*), together with the coordinates of the node are implemented via a call to the Tnode constructor. The drawing of the grid is obtained with a call to the *Draw* method on the Grid_pad object. The `G_EtaRange` and `G_PhiRange` are 40 (eta between -2 and +2) and 24 respectively.
The R-Phi grid – The grid in the left pad has been constructed using:
`TTUBS *tcons[EtaRange][PhiRange]`, in corrispondenza con la matrice
`TNode *tcnode [EtaRange][PhiRange],`
with `EtaRange=22` and `PhiRange=24`
(note that this part of the program has to be revised changing from the eta variable, which is not appropriate to a undistorted spatial representation, to θ). Here the basic elements are sections of cylindrical tubes, replacing the rectangular prism used for the eta-phi grid.
The R-phi display also includes an inner cylinder limiting the region available for tracking. The grid is constructed on a outer cylinder, corresponding to the inner face of the calorimeters (data in `Inner_Radius` and `Extern_Radius;` TCONS `*base_tube` (outer) and TCONS `*track_tube` (inner) are linked to the corresponding TNode `Bnode` and `Btrack_node)`. Both cylinders are drawn in the `TubeDraw(RPhi_pad)` method of EventDisplay, by using the TCons class.

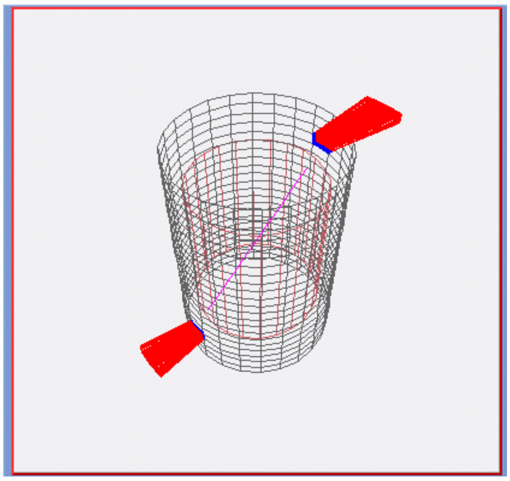


Figure 6.1 – The R-phi cylinders and the display of an “artificial” two tracks, back to back, event

6.3 Energy towers

The calorimetric data of the event are stored in two matrices:

```
Float_t energiaEm[G_EtaRange][G_PhiRange];
Float_t energiaHad[G_EtaRange][G_PhiRange];
```

The matrix elements are displayed as towers in the appropriate cell of the grids. The matching of matrix element and grid cell is performed by the `Fill_Eta_Phi method:`

```
Bool_t EventDisplay::Fill_Eta_Phi(Double_t eta, Double_t phi, int
                                   &IEta, int &IPhi){
      /* Sets the value of IEta and IPhi starting from eta and phi  */
      if(TMath::Abs(eta)>2) return 0;
      IEta=(int)TMath::Floor((eta+2)*10);
      if( (IEta<0)||(IEta>39) ) return 0;
      phi=phi*(180/TMath::Pi());
      (IPhi)=(int)TMath::Floor(phi/15);
      if((IPhi<0)||(IPhi>23)) return 0;
      return 1;
}
```

Before drawing the towers, a cleaning of the previous event is performed:

```
for(i=0;i<EtaRange;i++)
      for(j=0;j<PhiRange;j++){
        delete tc_heights[i][j];
        delete tc_heights2[i][j];
        delete tc_h_node[i][j];
        delete tc_h_node2[i][j];
      }
```

The `EventDisplay::TowerGenerator()`method is used to construct and draw the towers. For the Eta-Phi grid we use 4 matrices:

```
TPARA *heights[G_EtaRange][G_PhiRange]
TPARA *heights2[G_EtaRange][G_PhiRange];
```

Which, in a similar way to the cells of the grids, are linked to the matrices:

```
New_Node *h_node[G_EtaRange][G_PhiRange]
New_Node *h_node2[G_EtaRange][G_PhiRange]
```

The towers in the R-Phi grid are shaped as section of a tube. The height, proportional to the energy, is the thickness of the tube section (the difference between the external radius and the inner radius, the inner radius coinciding with the radius of the cylindrical grid). The matrices are:

```
TTUBS *tc_heights[EtaRange][PhiRange]
TTUBS *tc_heights2[EtaRange][PhiRange],
```

and they are linked to the matrices

```
New_Node *tc_h_node[EtaRange][PhiRange]
New_Node *tc_h_node2[EtaRange][PhiRange]
```

The heights of the em and had parts of the towers are defined by the tubes radii:

```
tc_heights[i][j]=new TTUBS("tc_Heights_i_j", "tc_Heights_i_j","void",
  Extern_Radius,Extern_Radius+param2*energiaEm[i+9][j], EtaStep,j*15,(j*15)+15);

tc_heights2[i][j]=new TTUBS("tc_Heights2_i_j", "tc_Heights2_i_j", "void",
  Extern_Radius+param2*energiaEm[i+9][j],
Extern_Radius+param2*energiaEm[i+9][j]+param2*energiaHad[i+9][j], EtaStep, j*15,
(j*15)+15);
```

6.4 – Tracks

The drawing of tracks is based on the THelix class of ROOT . The track constructor CDF_Helix uses the particle velocity (`Double_t v[3]`), initial position (`Double_t p[3]`), angular speed w, and range of the helix`(0,tmax)`. Here, speed and time interval are mnemonic names for the particle variables associated to the curvature of the trajectory. The time range is used to check that the trajectory of the particle is drawn up to the surface of the inner cylinder only. Every track created is put in the `HelixVector` array, which is reinitialized for every new event read.

6.5 – Interaction with towers and tracks

A useful feature of the display program is the possibility given to the user to interact with the displayed objects. By using the mouse, the user can select an object (tower or track) and get the information related to it. To obtain the interactivity, since the ExecuteEvent method of TNode is void, we had to redefine the method by extending the TNode class with New_Node. Then, "clicking" on the towers is enabled by the following procedure:

```
void New_Node::ExecuteEvent(Int_t event, Int_t px, Int_t py){

 switch(event){
 case kButton1Down: {.....
                      break;}
 case kMouseEnter:  {.....
                      break;}
 case kMouseLeave:  {.....
                      break;}
}
```

The New_Node constructor includes various parameters. A link to the *Edit* text area where the information has to be displayed and the Double_t vector *Infos*, which contains the numeric information of the selected object. The "events" *Button1Down*, *MouseEnter* and *MouseLeave* allow respectively to print in the text area the content of *Infos* and to change or reset the color of the selected object.