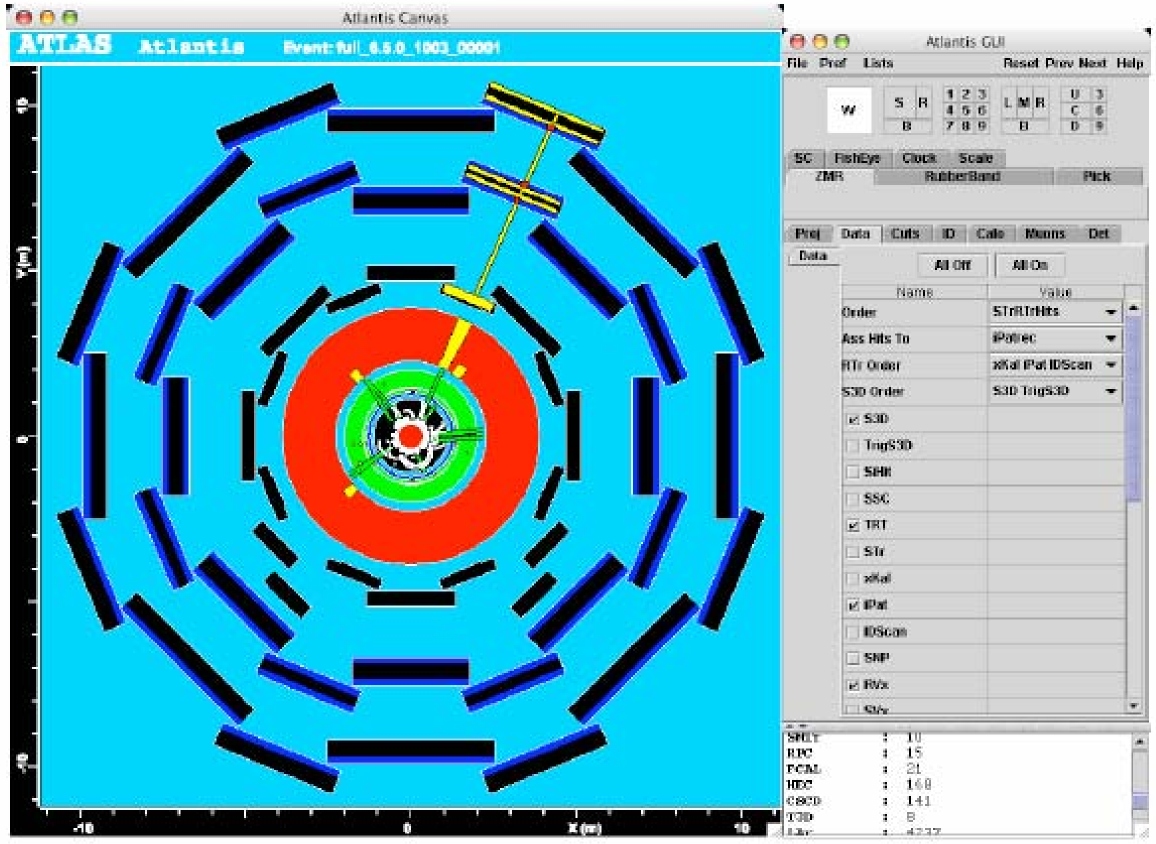
ATLAS Event Display

Henri Kowalski

4th of December 06, Hamburg

ATLANTIS



Name	Value
Order	STrRTrHits
Ass Hits To	iPatrec
RTr Order	xKal iPat IDScan
S3D Order	S3D TrigS3D
✓ S3D	
TrigS3D	
SHR	
SSC	
✓ TRT	
STr	
xKal	
✓ iPat	
IDScan	
SNP	
✓ RVx	
SVx	

```
SMDT     :   10
RPC      :   15
FCAL     :   21
HEC      :   168
CSCD     :   141
T3D      :   8
LAr      :   4737
```

Menue: I/O … Help

Interaction control

Pick, Zoom, Projections

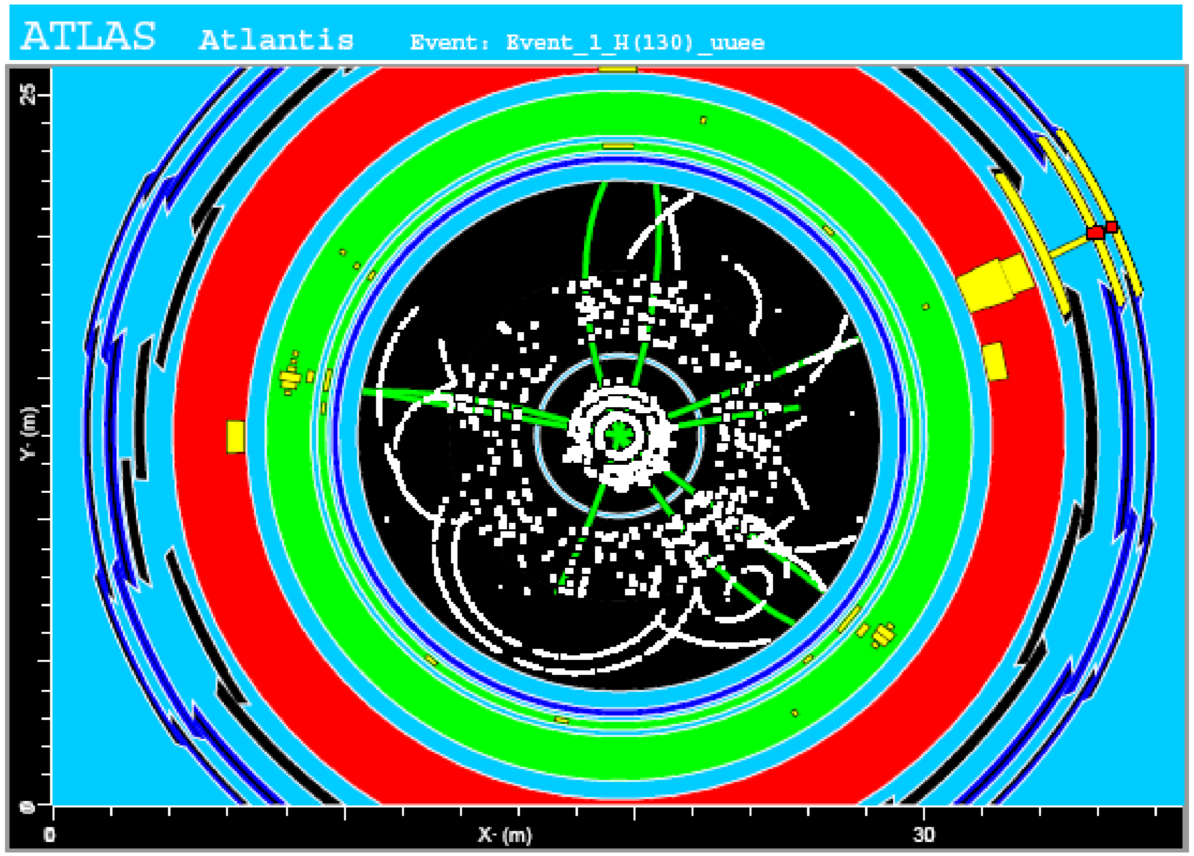
Parameter control

data selection, cuts,
subdetector systems,
projections

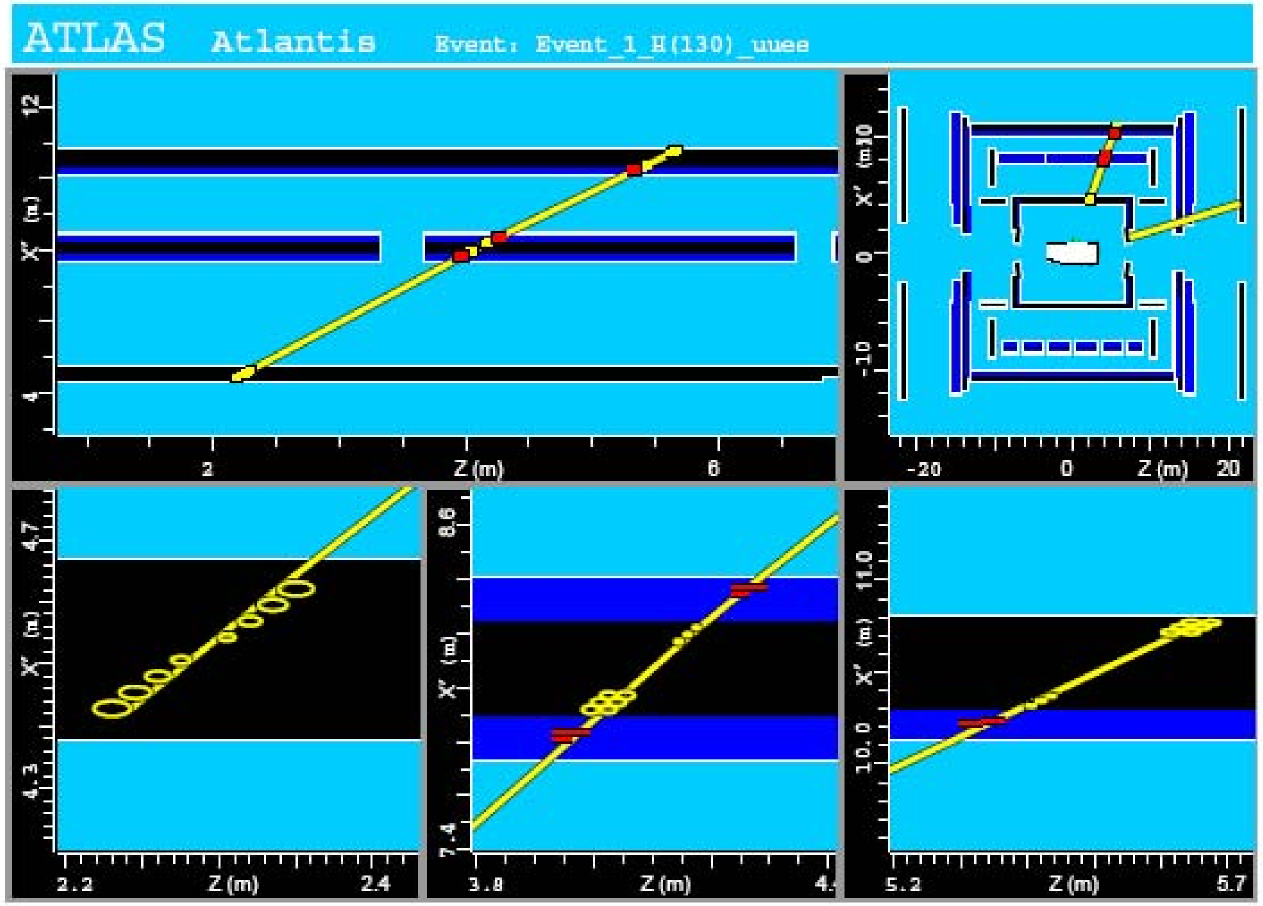
Output Display

canvas

Graphical User Interface
GUI

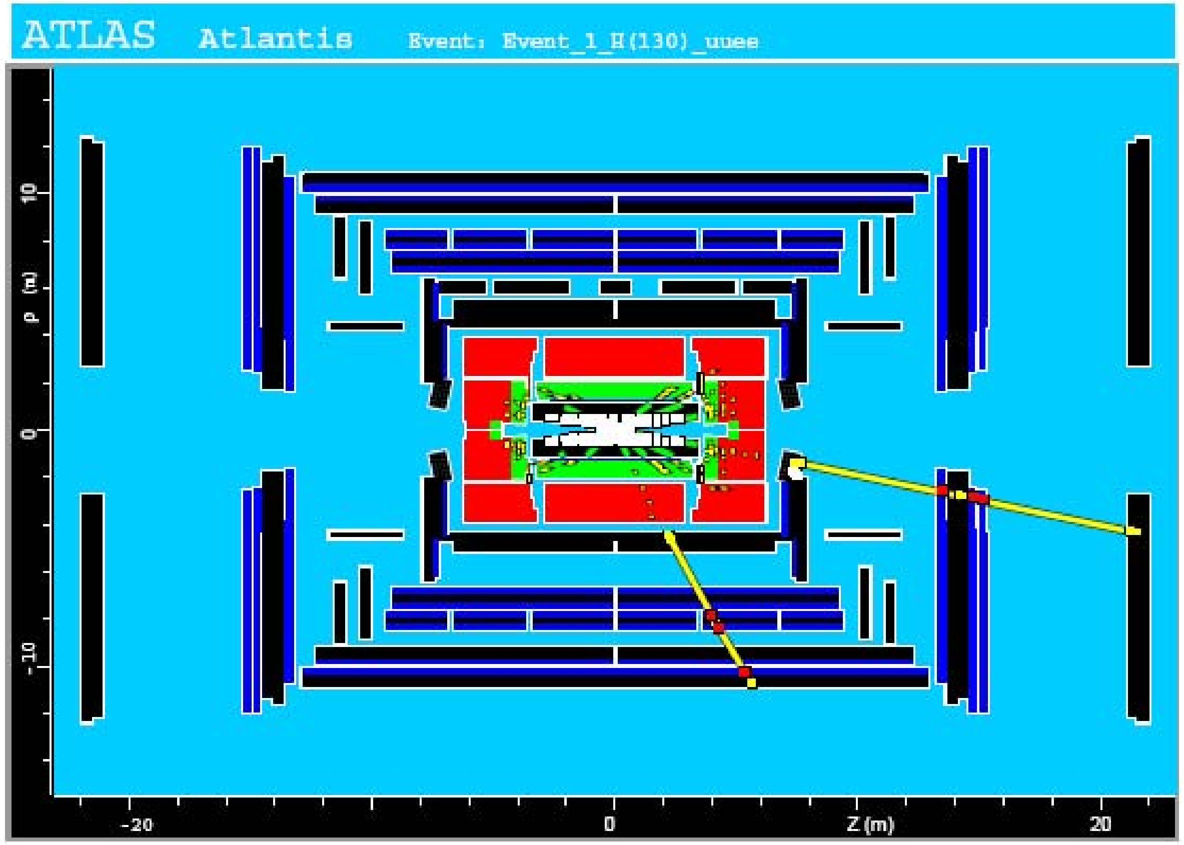


X/Y Projection with fisheye

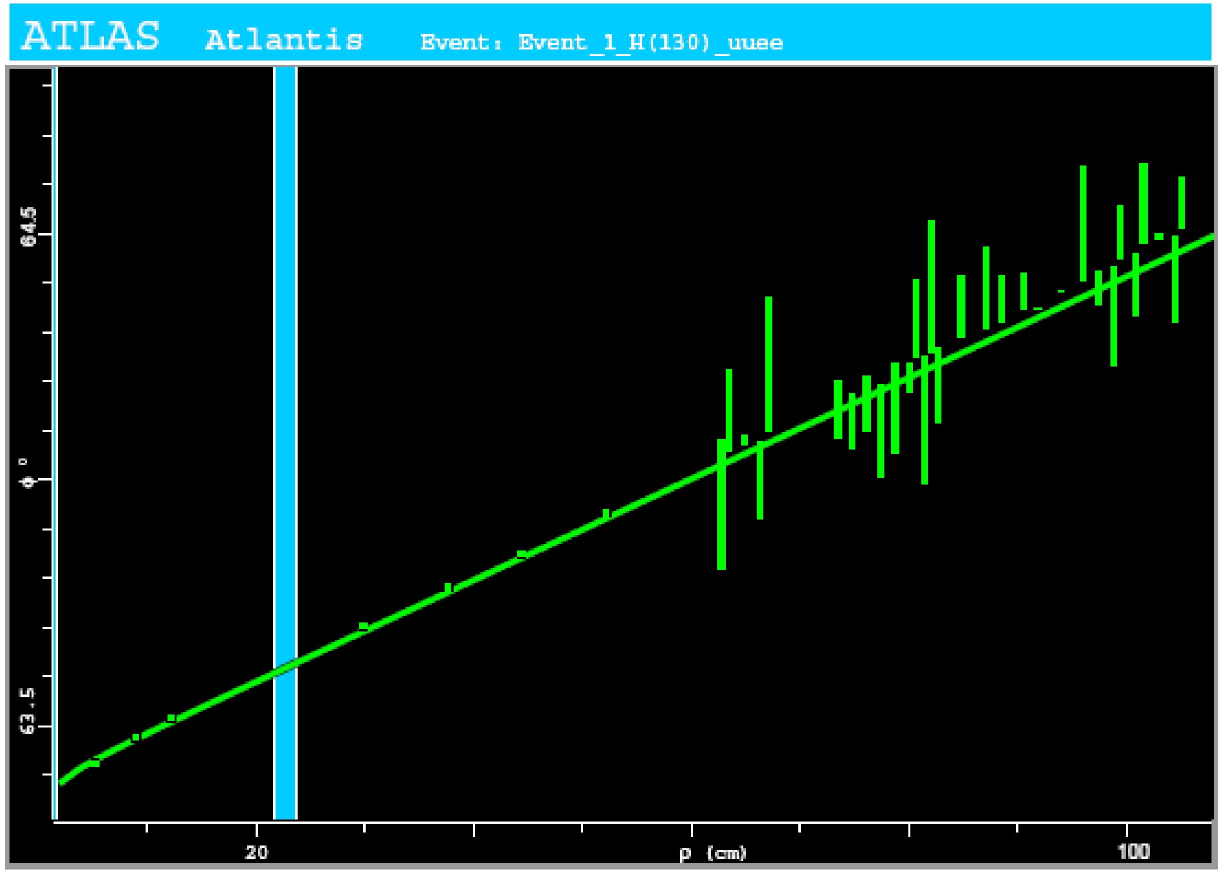


X'/Z Projection

Zoomed view of the uppermost muon track
Zoomed view for each muon MDT layer



ρ/Z Projection

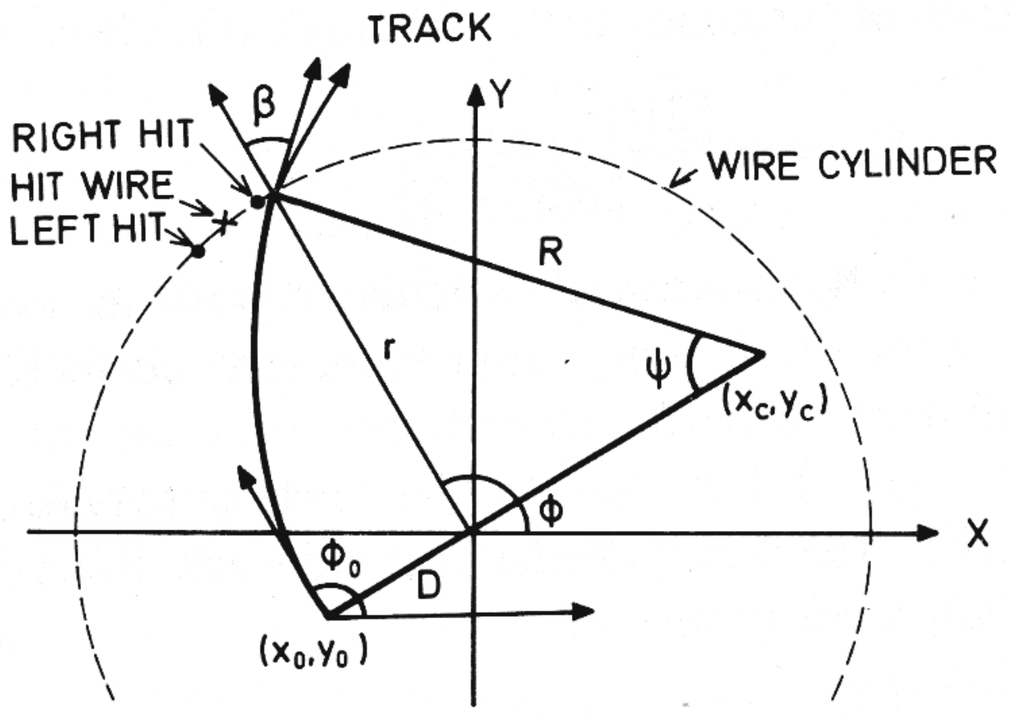


ϕ/ρ **Projection**
based on the relation:

$$\phi = \phi_0 + Q\frac{\rho}{2R}$$

Helix Equation in polar coordinates

see e.g. HK&DC
Nucl. Inst.
185 (1981) p 235

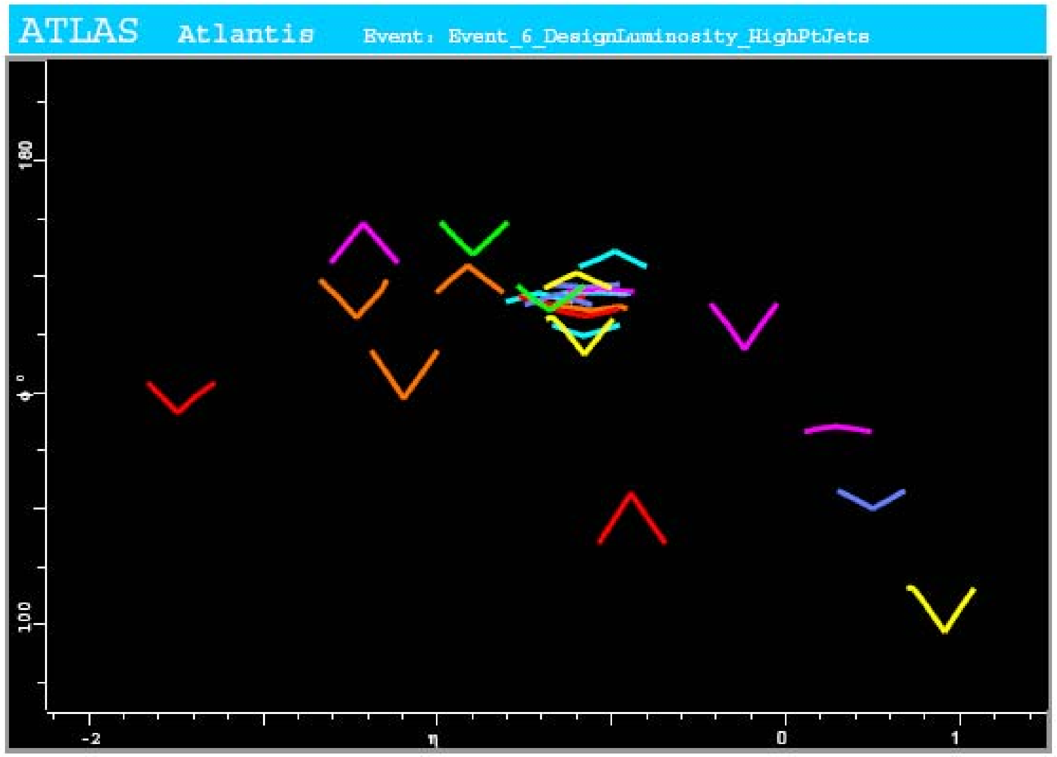


$$\rho = r$$

$$\phi = \phi_0 + Q \arcsin\left(\frac{2RD - D^2 - \rho^2}{2\rho(R-D)}\right)$$

$$D << R$$

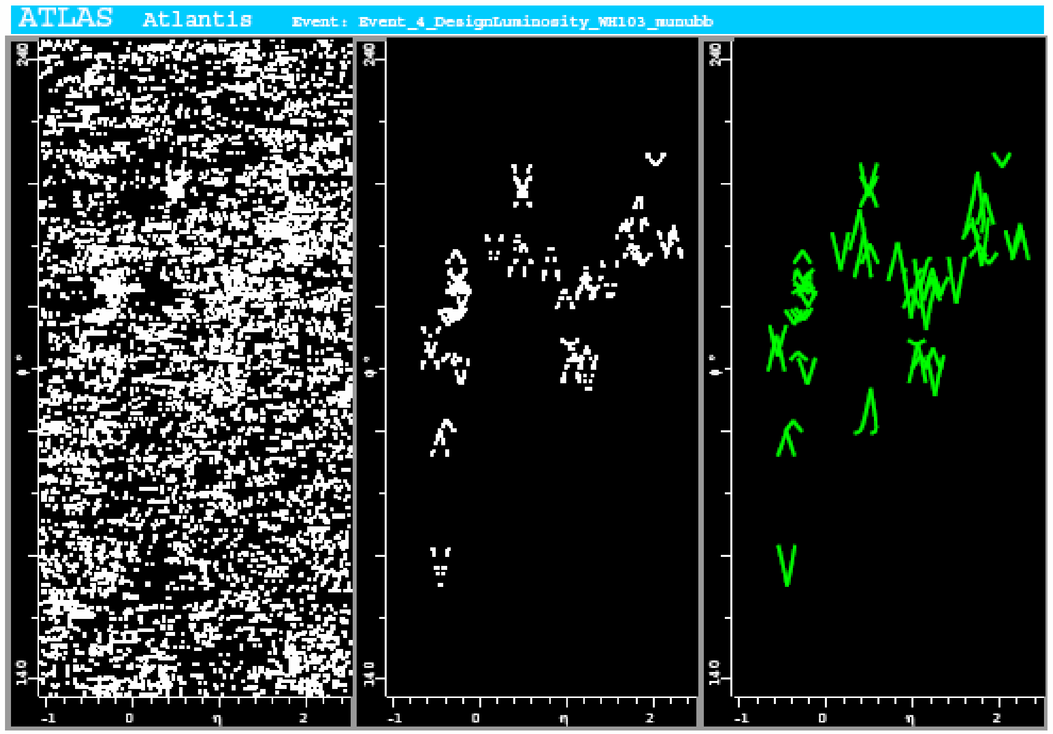
$$\Rightarrow \phi = \phi_0 + Q\frac{\rho}{2R}$$



The V-Plot
a combined ϕ/ρ and ϕ/η plot

$$(\phi,\ \eta \pm k \cdot (\rho_{MAX} - \rho))$$

V-Plot projections of a high luminosity event



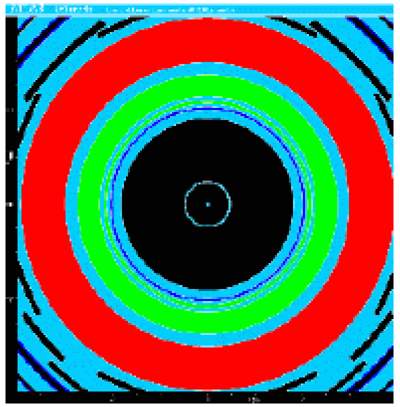
all hits

after filtering using ϕ,η,z plot

reconstructed tracks for comparison

In Atlantis data is read from XML files, produced by a dedicated algorithm JiveXML running within ATHENA

JiveXML



JiveXML is an Athena package that contains algorithms to convert event Data to XML files. The XML files can then be read in by the Atlantis Event Display. Both fully reconstructed and fast simulated events (Atlfast) can be converted to XML and viewed with Atlantis.

To display the maximum information from your events, you need to run the full reconstruction on data after digitization (e.g. using RecExCommon, see below). But JiveXML runs also on ESD files (which contain most reconstructed information). See some instructions from Tom LeCompte on how to run on AOD/ESD . ← Problem

Links

CVS Web
Atlantis
PPT Tutorial
JiveXML class diagram

Running JiveXML on AODs

Atlantis is an excellent display with some problems:

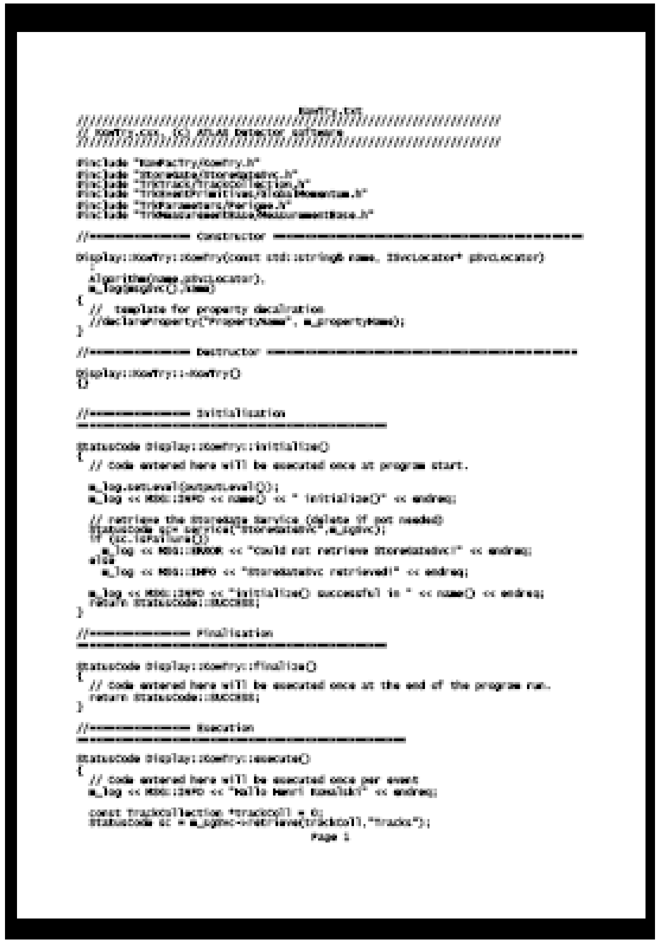
it is written in Java and not in C++
➔ XML interface is necessary
JiveXML has to be recreated after any change of code,
full compilation of the reconstruction code is very difficult
possible solution:
create JiveXML for every software package independently

the logic of data navigation in Atlantis does not correspond to
the logic of the reconstruction software (Atlas C++)
➔ very limited use for debugging problems

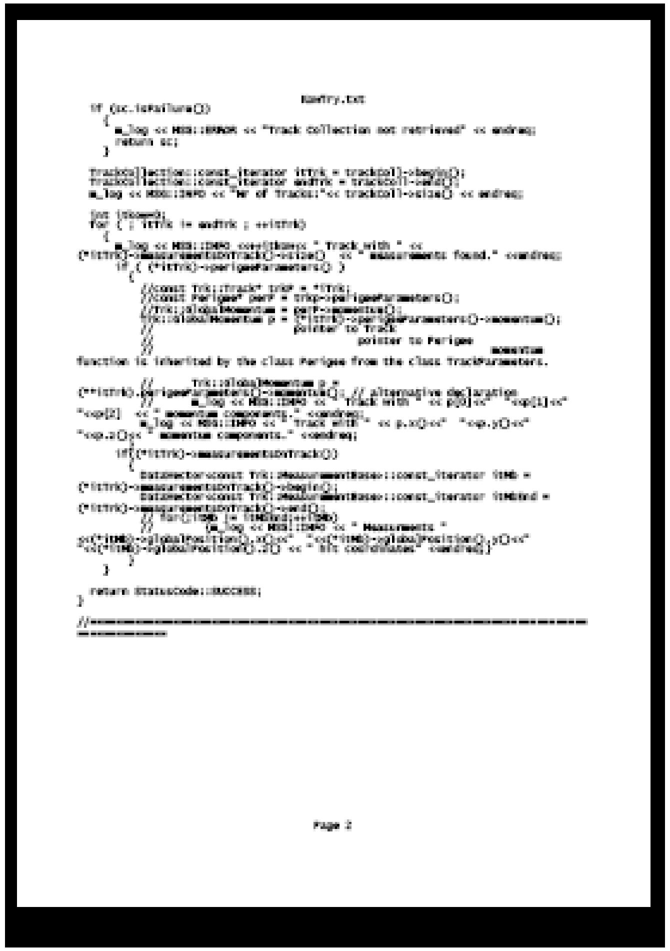
Atlantis is a 2D display ➔ limited use for muons

ATLAS Software
C++

Exercise: Print out tracks and their hits (with help by M. Siebel)



```
                                KowTry.txt
/////////////////////////////////////////////////////////////////////////////
// KowTry.cxx, (c) ATLAS Detector software
/////////////////////////////////////////////////////////////////////////////

#include "KowFactory/KowTry.h"
#include "StoreGate/StoreGateSvc.h"
#include "TrkTrack/TrackCollection.h"
#include "TrkEventPrimitives/GlobalMomentum.h"
#include "TrkParameters/Perigee.h"
#include "TrkMeasurementBase/MeasurementBase.h"

//================ Constructor =================================================

Display::KowTry::KowTry(const std::string& name, ISvcLocator* pSvcLocator)
  :
  Algorithm(name,pSvcLocator),
  m_log(msgSvc(),name)
{
  //  template for property decalration
  //declareProperty("PropertyName", m_propertyName);
}

//================ Destructor =================================================

Display::KowTry::~KowTry()
{}


//================ Initialisation
=================================================

StatusCode Display::KowTry::initialize()
{
  // Code entered here will be executed once at program start.

  m_log.setLevel(outputLevel());
  m_log << MSG::INFO << name() << " initialize()" << endreq;

  // retrieve the StoreGate Service (delete if not needed)
  StatusCode sc= service("StoreGateSvc",m_sgSvc);
  if (sc.isFailure())
    m_log << MSG::ERROR << "Could not retrieve StoreGateSvc!" << endreq;
  else
    m_log << MSG::INFO << "StoreGateSvc retrieved!" << endreq;

  m_log << MSG::INFO << "initialize() successful in " << name() << endreq;
  return StatusCode::SUCCESS;
}

//================ Finalisation
=================================================

StatusCode Display::KowTry::finalize()
{
  // Code entered here will be executed once at the end of the program run.
  return StatusCode::SUCCESS;
}

//================ Execution
====================================================

StatusCode Display::KowTry::execute()
{
  // Code entered here will be executed once per event
  m_log << MSG::INFO << "Hallo Henri Kowalski" << endreq;

  const TrackCollection *trackColl = 0;
  StatusCode sc = m_sgSvc->retrieve(trackColl,"Tracks");
                                  Page 1
```



```
                                KowTry.txt
  if (sc.isFailure())
    {
      m_log << MSG::ERROR << "Track Collection not retrieved" << endreq;
      return sc;
    }

  TrackCollection::const_iterator itTrk = trackColl->begin();
  TrackCollection::const_iterator endTrk = trackColl->end();
  m_log << MSG::INFO << "Nr of Tracks:"<< trackColl->size() << endreq;

  int itCount;
  for ( ; itTrk != endTrk ; ++itTrk)
    {
      m_log << MSG::INFO <<<<itCount<< " Track with " <<
(*itTrk)->measurementsOnTrack()->size()  << " measurements found." <<endreq;
      if ( (*itTrk)->perigeeParameters() )
       {
         //const Trk::Track* trkP = *itTrk;
         //const Perigee* perP = trkP->perigeeParameters();
         //Trk::GlobalMomentum = perP->momentum();
         Trk::GlobalMomentum p = (*itTrk)->perigeeParameters()->momentum();
         //                      pointer to Track
         //                                        pointer to Perigee
         //                                                          momentum
function is inherited by the class Perigee from the class TrackParameters.

         //         Trk::GlobalMomentum p =
(**itTrk).perigeeParameters()->momentum(); // alternative declaration
         //         m_log << MSG::INFO << " Track with " << p[0]<<"  "<<p[1]<<"
"<<p[2]  << " momentum components." <<endreq;
         m_log << MSG::INFO << " Track with " << p.x()<<"  "<<p.y()<<"
"<<p.z()<< " momentum components." <<endreq;
       }
      if((*itTrk)->measurementsOnTrack())
       {
         DataVector<const Trk::MeasurementBase>::const_iterator itMB =
(*itTrk)->measurementsOnTrack()->begin();
         DataVector<const Trk::MeasurementBase>::const_iterator itMBEnd =
(*itTrk)->measurementsOnTrack()->end();
         // for(;itMB != itMBEnd;++itMB)
         //            {m_log << MSG::INFO << " Measurements "
<<(*itMB)->globalPosition().x()<<"  "<<(*itMB)->globalPosition().y()<<"
"<<(*itMB)->globalPosition().z() << " hit coordinates" <<endreq;}
       }
    }

  return StatusCode::SUCCESS;
}

//=============================================================================
=============
                                  Page 2
```

Exercise: Print out tracks and their hits

Retrieve tracks from the StoreGate

```
const TrackCollection *trackColl = 0;
StatusCode sc = m_sgSvc->retrieve(trackColl,"Tracks");
```

Find pointers to the beginning and end of the track container

```
TrackCollection::const_iterator itTrk = trackColl->begin();
TrackCollection::const_iterator endTrk = trackColl->end();
m_log << MSG::INFO << "Nr of Tracks:"<< trackColl->size() << endreq;
```

Set the loop over tracks

```
 int itkow=0;
 for ( ; itTrk != endTrk ; ++itTrk)
   {
     m_log << MSG::INFO <<++itkow<< " Track with " <<
(*itTrk)->measurementsOnTrack()->size()   << " measurements found." <<endreq;
```

Access momentum of the track

```
    if ( (*itTrk)->perigeeParameters() )
      {
         Trk::GlobalMomentum p = (*itTrk)->perigeeParameters()->momentum();
         //                      pointer to Track
         //                                 pointer to Perigee
         //                                                       momentum
function is inherited by the class Perigee from the class TrackParameters.
```

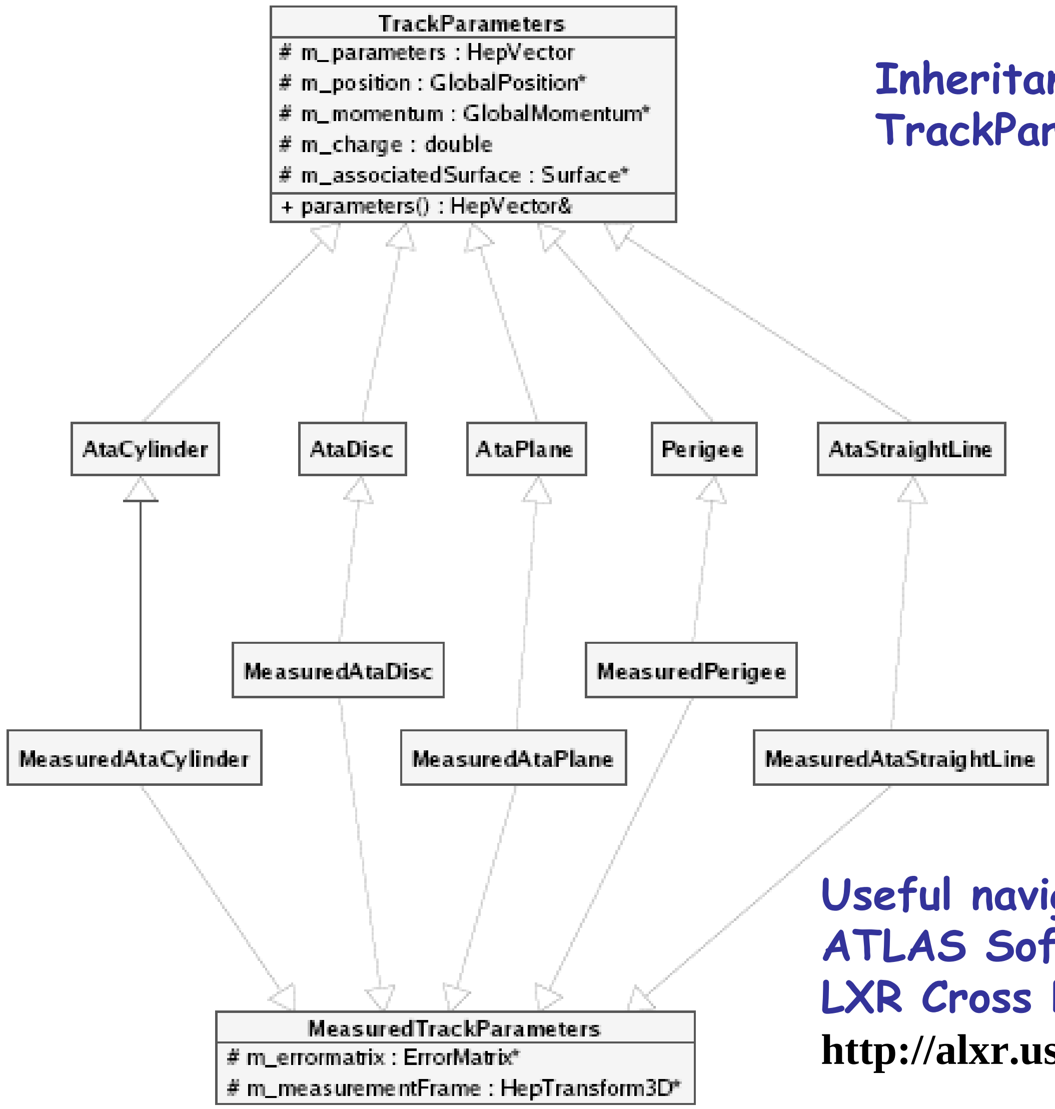
Print out of the momentum

```
           m_log << MSG::INFO << " Track with " << p.x()<<"  "<<p.y()<<"
 "<<p.z()<< " momentum components." <<endreq;
         }
```

Access and printout of hits belonging to the track

```
      if((*itTrk)->measurementsOnTrack())
        {
          DataVector<const Trk::MeasurementBase>::const_iterator itMb =
 (*itTrk)->measurementsOnTrack()->begin();
          DataVector<const Trk::MeasurementBase>::const_iterator itMbEnd =
 (*itTrk)->measurementsOnTrack()->end();
          // for(;itMb != itMbEnd;++itMb)
          //          {m_log << MSG::INFO << " Measurments "
 <<(*itMb)->globalPosition().x()<<"  "<<(*itMb)->globalPosition().y()<<"
 "<<(*itMb)->globalPosition().z() << " hit coordinates" <<endreq;}
        }
    }
```

ATLAS Tracking Event Data Model


TrackParameters
# m_parameters : HepVector
# m_position : GlobalPosition*
# m_momentum : GlobalMomentum*
# m_charge : double
# m_associatedSurface : Surface*
+ parameters() : HepVector&
AtaCylinder
AtaDisc
AtaPlane
Perigee
AtaStraightLine
MeasuredAtaDisc
MeasuredPerigee
MeasuredAtaCylinder
MeasuredAtaPlane
MeasuredAtaStraightLine
MeasuredTrackParameters
# m_errormatrix : ErrorMatrix*
# m_measurementFrame : HepTransform3D*

Inheritance structure of TrackParameters data classes

Useful navigation tool through ATLAS Software:
LXR Cross Referencer
http://alxr.usatlas.bnl.gov/lxr/source

Exercise: Match tracks with calorimeter cells (help by M. Siebel)

```
double distbar = 0.;
double distec = 0.;
CaloCell_ID::CaloSample sample;
```

Input: trketa Output: distbar - distance to barrel

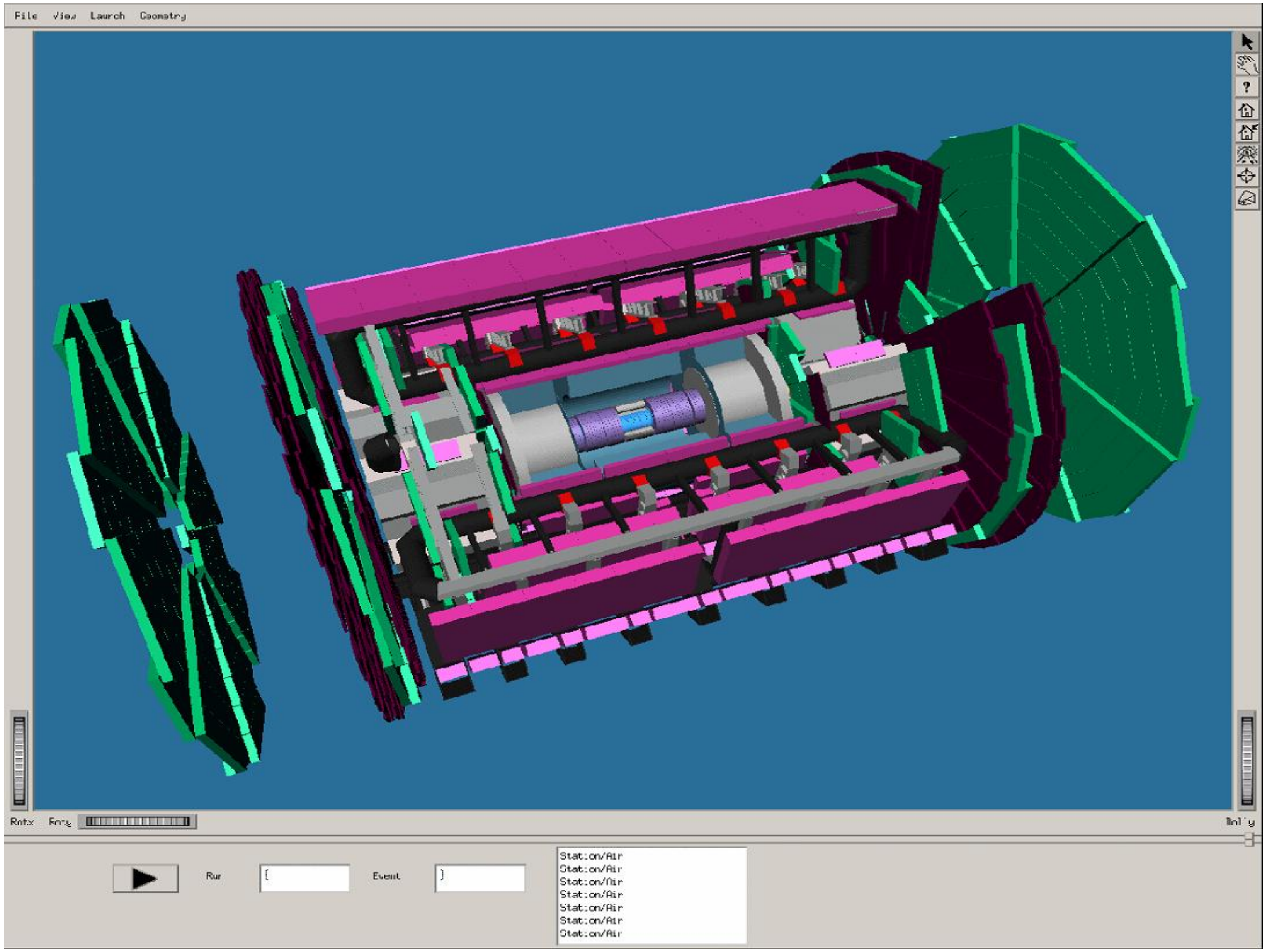
distec - distance to endcap

```
// PS :
distbar = m_calodepth->deta(CaloCell_ID::PreSamplerB,trketa);
distec = m_calodepth->deta(CaloCell_ID::PreSamplerE,trketa);
```

Get sample – ID of the calorimeter cell

```
    // middle :
    distbar = m_calodepth->deta(CaloCell_ID::EMB2,trketa);
    distec = m_calodepth->deta(CaloCell_ID::EME2,trketa);

    log << MSG::DEBUG << " TrackTo ...Middle : for eta= " << trketa << " dist to
Barrel =" << distbar
        << " to endcap =" << distec << endreq;
    if (distbar < 0 ) sample = CaloCell_ID::EMB2;
```

V-Atlas

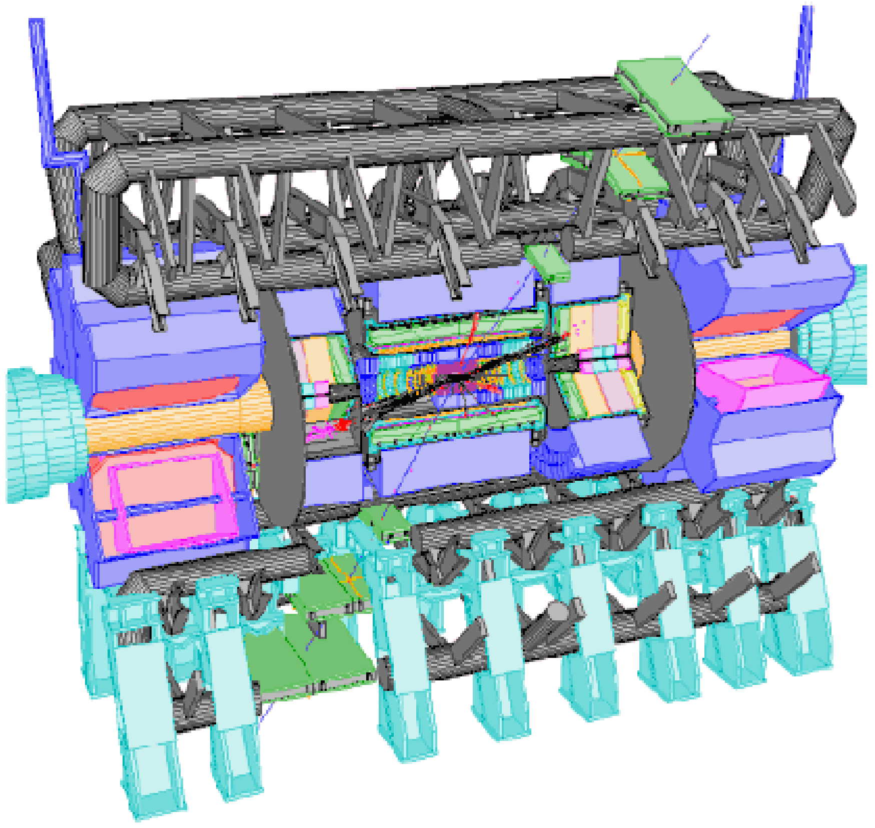


Station/Air
Station/Air
Station/Air
Station/Air
Station/Air
Station/Air
Station/Air

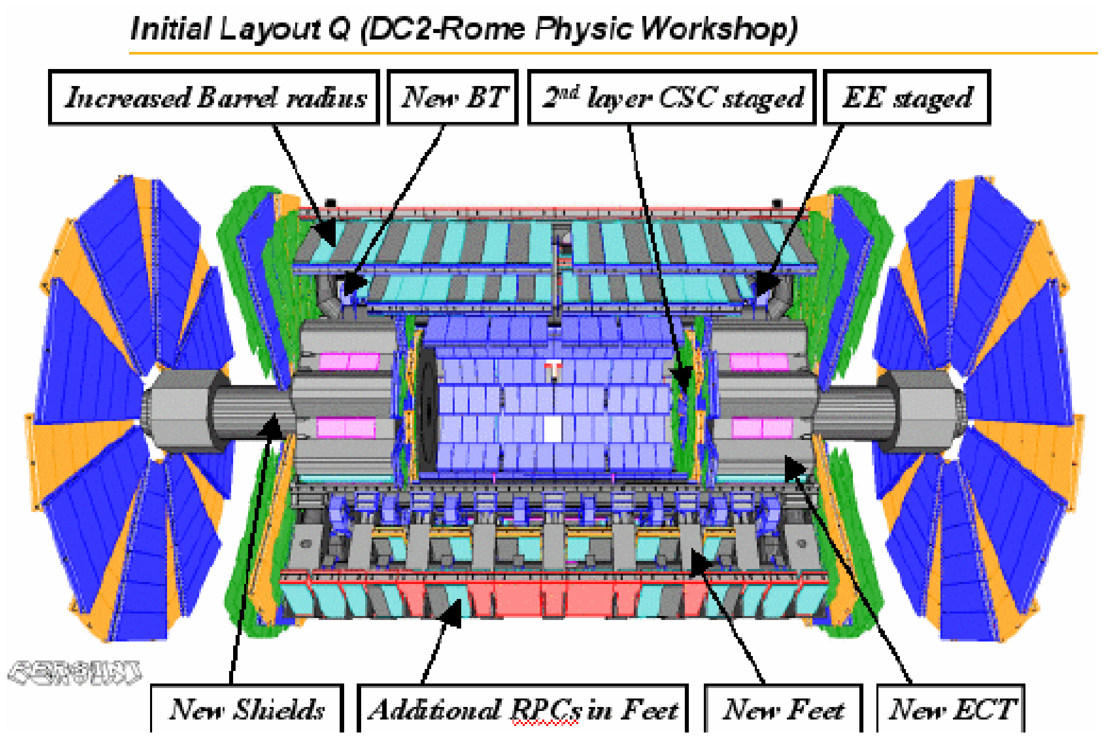
V-Atlas

- V-Atlas is the Event Visualization program integrated into ATLAS analysis framework ATHENA
- V-Atlas is based upon **Open Inventor** and it’s **HEPVis** extensions
- V-Atlas co-displays the real Detector Description/Simulation geometry together with event data
- V-Atlas renders in real time on regular laptop computers, using their available graphics acceleration.
- No commercial software is required
- V-Atlas has been also actively used as a powerful debugging tool in various domains of ATLAS s/w

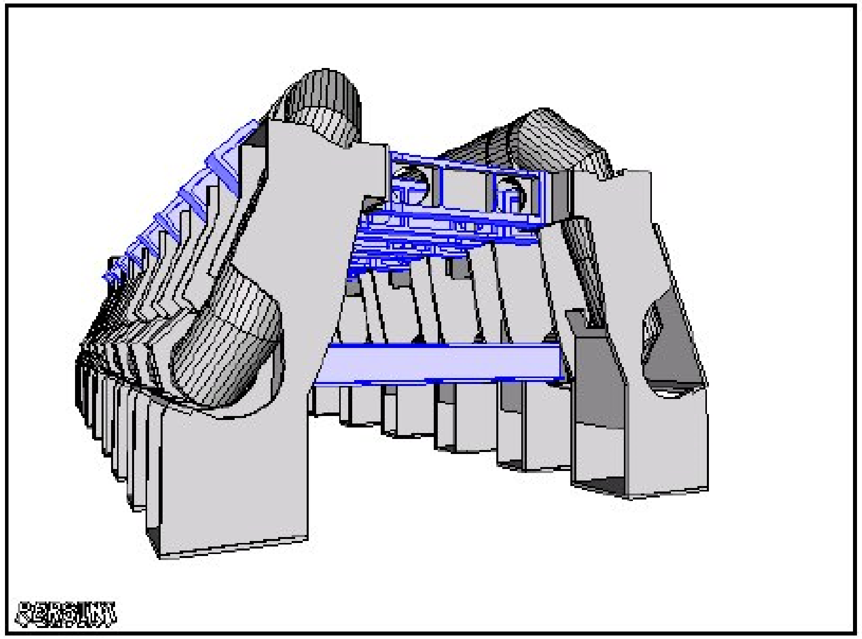
PERSINT
Perspectively Interacting

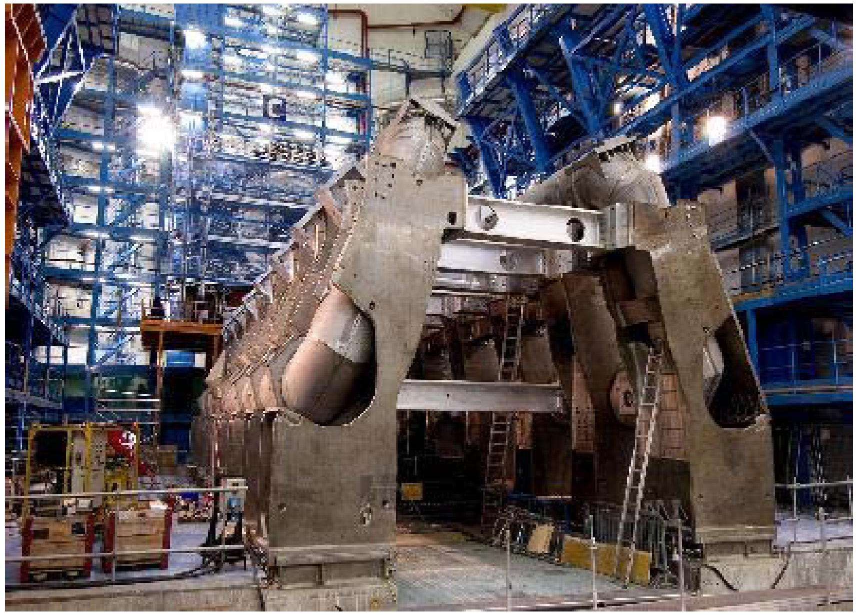


PERSINT is a 3D package designed by Marc Virchaux for debugging the muon reconstruction code and the detector geometry description



Accurate measurements of the muon tracks in a complicated magnetic field require a 3D graphics tool





PERSINT has a very nice 3D graphics
serious problem - it is written in FORTRAN90

Useful link:
https://twiki.cern.ch/twiki/bin/view/Atlas/DetDescAndGraphicsReview