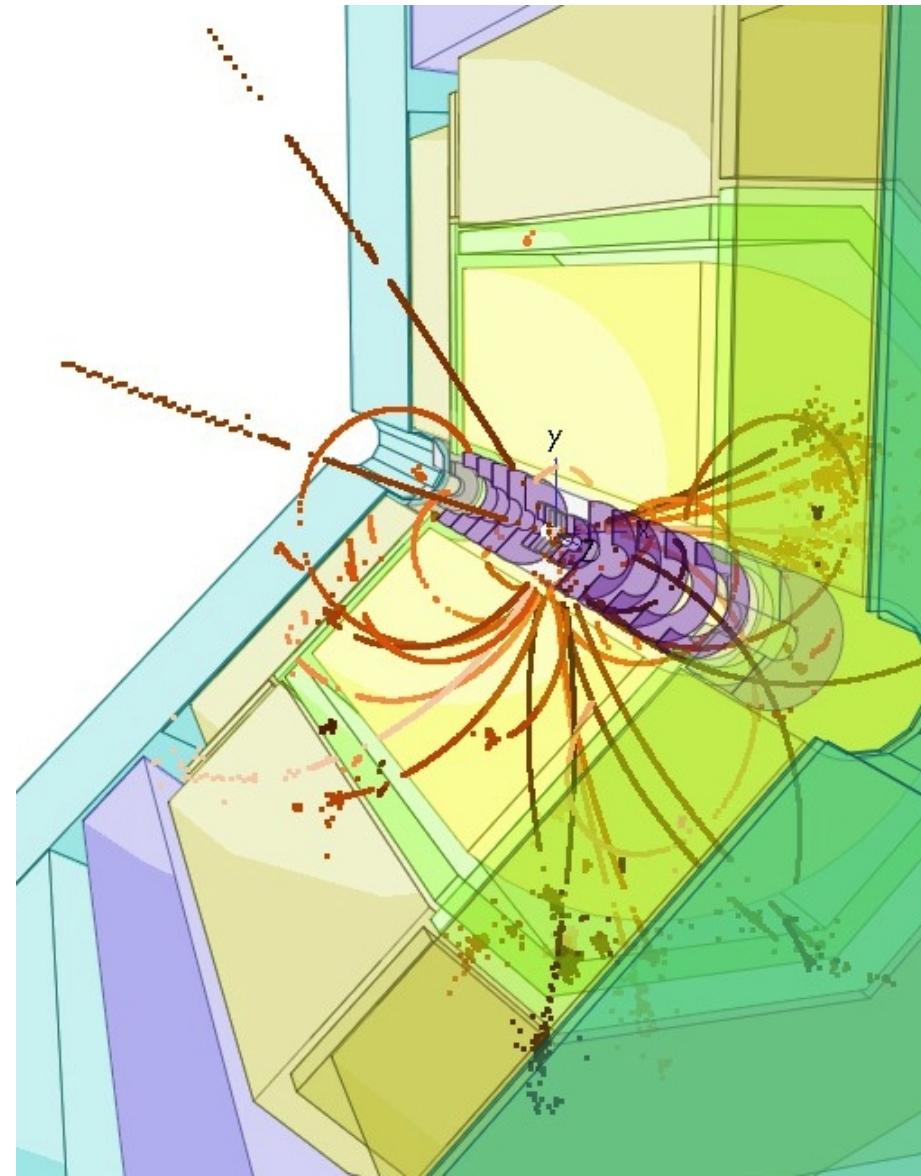


Technical aspects of High Level Reconstruction Tools

Frank Gaede
High Level Reconstruction Week
July 6-10, DESY, 2015

Outline

- general remarks
- using ParticleID objects
- mini-DSTs



general remarks

- ILD has a standard reconstruction defined for the DBD (already for LOI):
 - unique collection of **Tracks**: **MarlinTrkTracks**
 - unique collection of **Clusters**: **PandoraClusters**
 - combined into unique collection of **ReconstructedParticles**: **PandorPFOs**
- => looping over PandorPFOs gives best estimate of the event (no double counting or missing E, except for BeamCal)
- High level reconstruction tools are essentially those that run on PandorPFOs (possibly using Track information):
 - vertex finders, Jet finders, flavor tagging, Isolated Lepton Finders,....

...general remarks

- for these HLR tools it is safe to run **multiple algorithms** that possibly provide different or even contradicting hypothesis, depending on the requirements of the individual analysis
 - can have multiple jet collections in parallel or add arbitrary #PID objects to every ReconstructedParticle (see later)
- these HLR tools can essentially be run on DSTs
 - exceptions: dEdx and shower shapes processor need to be run on the hits
- more difficult: alternative or refined 'low level' reconstruction algorithms, like Photon finders(Garlic), VO or muon finders
- -> **possible double counting** if run in parallel to 'standard reconstruction'
 - need to discuss at this workshop how to best combine the best tools into **one consistent reconstruction** chain or allow for a parallel **alternative reconstruction** (PFO collection)

ParticleID

virtual int **getType () const** =0

Type - userdefined.

?

virtual int **getPDG () const** =0

The PDG code of this id - UnknownPDG (999999) if unknown.

virtual float **getLikelihood () const** =0

The likelihood of this hypothesis - in a user defined normalization

=> one PID object
per PID hypothesis

virtual int **getAlgorithmType () const** =0

Type of the algorithm/module that created this hypothesis - NOTE: must be unique within one collection.

virtual const **FloatVec &** **getParameters () const** =0

Parameters associated with this hypothesis.

virtual const **ParticleIDVec &** **getParticleIDs () const** =0

The particle Id's sorted by their likelihood.

virtual **ParticleID *** **getParticleIDUsed () const** =0

The particle Id used for the kinematics of this particle.

virtual float **getGoodnessOfPID () const** =0

The overall goodness of the PID on a scale of [0;1].

?

- ReconstructedParticles have vector of ParticleID objects
- can in principle be accessed directly - better **ParticleIDHandler**

PIDHandler API

PIDHandler (const LCCollection *col)

Create **PIDHandler** for reading from the given collection - read the collection parameters PIDAlgorithmTypeName, PIDAlgorithmTypeID and the corresponding ParameterNames_PIDAlgorithmTypeName if they exist.

PIDHandler (LCCollection *col)

Create **PIDHandler** for setting PID information in the given collection.

~PIDHandler ()

Update the collection parameter when going out of scope.

int addAlgorithm (const std::string &algoName, const StringVec ¶meterNames)

Add a new algorithm and return the corresponding algorithm id.

int getAlgorithmID (const std::string &algoName)

Return the typeID of algorithm algoName - throws **UnknownAlgorithm**.

const std::string & getAlgorithmName (int algoid)

Return the name of the algorithm with id - throws **UnknownAlgorithm**.

int getParameterIndex (int algorithmID, const std::string &pName)

The index of parameter pName for the algorithm with algorithmID - throws UnknownAlgorithm.

const ParticleID & getParticleID (LCOObject *particle, int algorithmID)

Return the (first) ParticleID object for the given algorithm and particle (or cluster) - throws **UnknownAlgorithm**.

ParticleIDVec getParticleIDs (LCOObject *p, int id)

Return all PID objects for a given algorithm - ordered with decreasing likelihood - t

recently added

void setParticleIDUsed (IMPL::ReconstructedParticleImpl *particle, int algorithmID)

Set the particleID algorithm that is used for this particle's kinematic variables.

const StringVec & getParameterNames (int algorithmID)

The names of parameters for the algorithm with algorithmID - throws UnknownAlgorithm.

const IntVec & getAlgorithmIDs ()

Return the IDs of all known Algorithms.

void setParticleID (LCOObject *p, int userType, int PDG, float likelihood, int algorithmID, const FloatVec ¶ms)

Set the particleID information for this particle (or cluster) - throws **UnknownAlgorithm**.

example: writing PID information

```
LCCollection* col = evt->getCollection("PandoraPFOs")  
PIDHandler pidHand( col ) ;  
myID = pidHand( "dEdx" , StringVec() ) ; // no extra params  
for( unsigned i=0,n=col->size() ; i<n ; ++i){  
    // compute dEdx likelyhoods  
    //...  
    pidHand.setParticleID( rec, 0, 13, Pmuon , myID, FloatVec() ) ;  
    pidHand.setParticleID( rec, 0, 311, Pkaon , myID, FloatVec() ) ;  
    pidHand.setParticleID( rec, 0, 211, Ppion , myID, FloatVec() ) ;  
}  
// one PID object per hypothesis  
//NB: PID objects can be written also on "Read-Only" collections
```

example: reading PID information

```
LCCollection* col = evt->getCollection("PandoraPFOs")  
  
PIDHandler pidHand( col ) ;  
  
myID = pidHand.getAlgorithmID( "dEdx" ) ;  
  
for( unsigned i=0,n=col->size() ; i<n ; ++i){  
  
    ReconstructedParticle* rec = ... col->getElementAt(i) ;  
  
    dedxPIDs = pidHand( rec, myID ) ;  
  
    for( unsigned j=0,m=dedxPIDs->size() ; j<m ; ++j){  
  
        std::cout << " pdg " << dedxPIDs[j].getPDG()  
                  << " likelihood " << dedxPIDs[j].getLikelihood()  
                  << std::endl ;  
  
    }  
  
    // NB: pid objects sorted wrt likelihood ...  
  
    // ... might not be the most convenient choice - see next slide
```

'alternative' use of ParticleID

```
PIDHandler *pidh= new PIDHandler(pfcol); //register PFO collection to PIDHandler  
lcio::ReconstructedParticle *pfo; //get PFO particles  
pidh->getParticleID(pfo, pidh->getAlgorithmID("LikelihoodPID")).getPDG(); //get absolute PDG value(no charge)  
pidh->getParticleID(pfo, pidh->getAlgorithmID("LikelihoodPID")).getLikelihood(); //get posterior prob.  
pidh->getParticleID(pfo, pidh->getAlgorithmID("LikelihoodPID")).getParameters()[0]; //electron hypothesis  
pidh->getParticleID(pfo, pidh->getAlgorithmID("LikelihoodPID")).getParameters()[1]; //muon hypothesis  
pidh->getParticleID(pfo, pidh->getAlgorithmID("LikelihoodPID")).getParameters()[2]; //pion hypothesis  
pidh->getParticleID(pfo, pidh->getAlgorithmID("LikelihoodPID")).getParameters()[3]; //kaon hypothesis  
pidh->getParticleID(pfo, pidh->getAlgorithmID("LikelihoodPID")).getParameters()[4]; //proton hypothesis
```

- example from M.Kurata
- simply use free parameters from ParticleID object to store a fixed number of PID likelihoods
- possibly easier to use !?
- could use parameter naming mechanism to make access more generic (not having to rely on hardcoded numbers)
- should agree on **one canonical way of how to use the ParticleIDs** - and then properly document it
- open for discussion

mini-DSTs

- suggestion to have a smaller file set for analyses, where Track and Cluster collections are dropped (mini-DSTs)
- pro
 - less disk space
 - faster reading
- con
 - would miss information from Tracks and Clusters (d0, #hits, position,...)
- possible solutions:
- add new **PFOTrkCluSummary** class to LCIO EDM
 - use this to summarize the information that would be needed from Tracks and Clusters
- extend **LCTuple** to write such information and create smaller Root ntuples/trees