Clustering approach to TPC pattern recognition

Frank Gaede, DESY

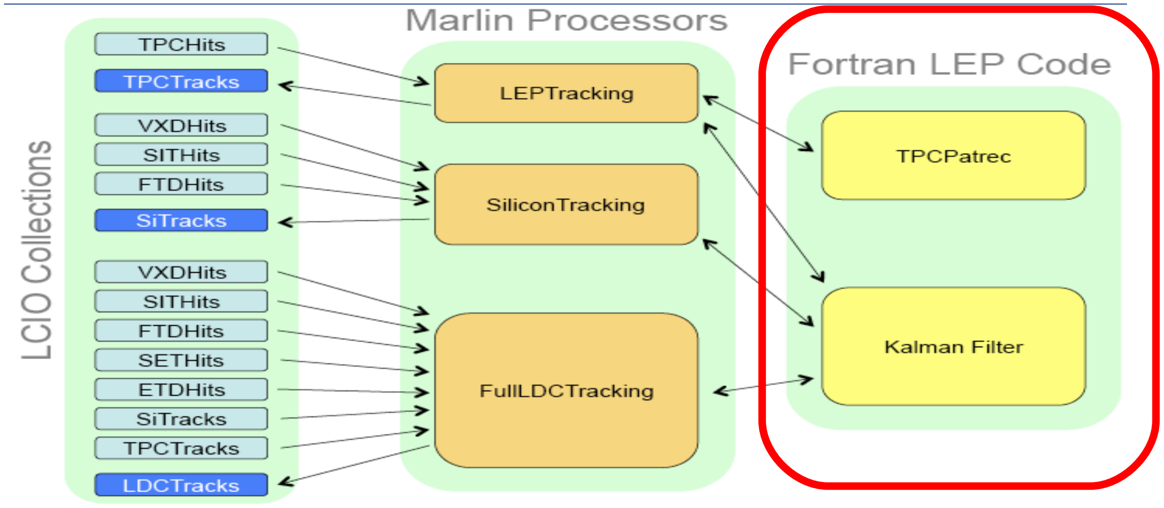
International Workshop on Linear Colliders

CERN, Oct. 18-22, 2010

Outline

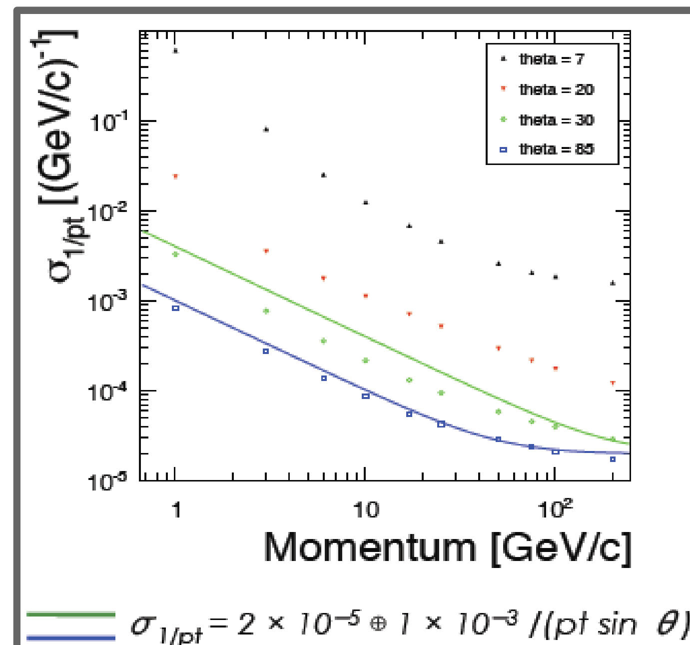
- Motivation
- KalTest Kalman Fitter
- Clustering algorithm
- TPC pattern recognition with clustering
- first 'results'
- Summary Outlook

ILD Tracking software



- standalone tracking in TPC
  - LEP code (f77)
- standalone patrec in VXD/SIT/FTD – in Marlin (C++)
- merging of Track segments and refit w/ f77 Kalman fitter

- current tracking used for LOI process
  - required p_t resolution reached
  - also in presence of backgrounds (even bg*3)
- issues:
  - f77: maintenance 'nightmare' !
  - homogeneous B-field only !
  - difficult to use with backgrounds
  - only single BX reconstruction
  - issues at 1-3 TeV
  - no strip tracking (ghost hits)



- -> need for a new tracking package

new Tracking code for iLCSoft

- had a look into ATLAS tracking code (S.Aplin)
- full featured modern PatRec:
  - (combinatorical) Kalman Filter
  - Gaussian Sum Filter, DAF,...
  - modular design
- hoped for simple integration into Marlin - however
  - rather tight coupling to GAUDI and Athena frameworks
    - algtools, DataVec, logging,...
- too involved for now

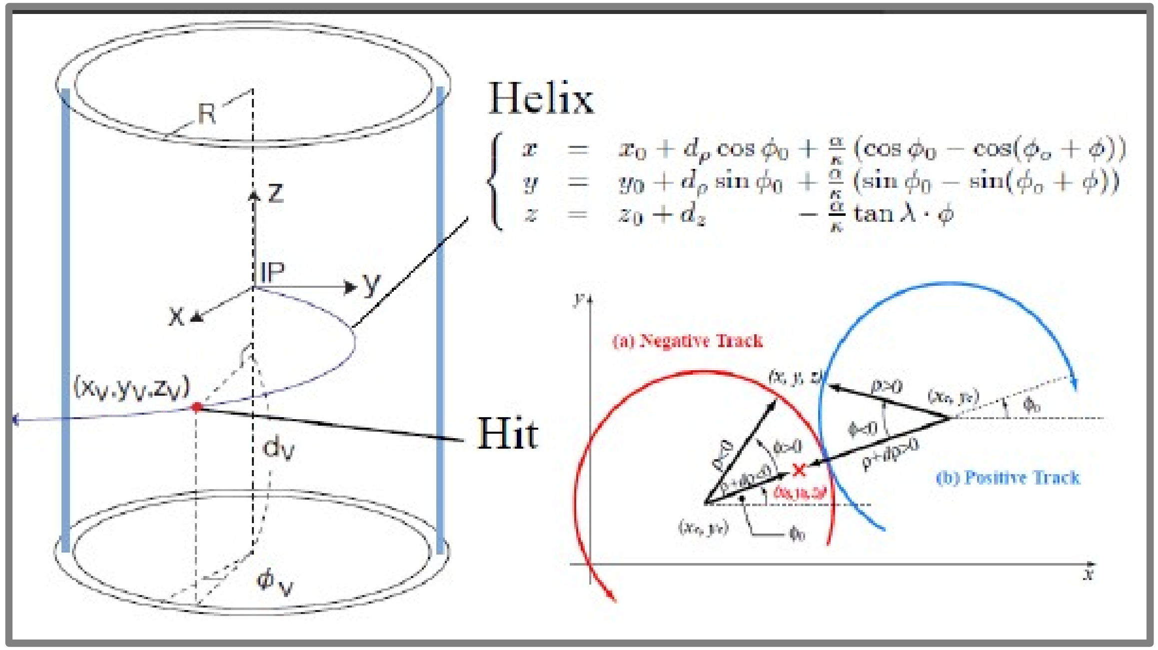
- also checked other Tracking/Fitting packages:
  - KalTest
    - developed @ KEK, used by LCTPC, based on ROOT
  - GenFit
    - developed @ TU Munich, to be used for SuperBelle, ROOT based
  - both seem to be good candidates for developing a new iLCSoft tracking package
  - start with evaluating KalTest
    - develop independent TPC patrec
    - use KalTest for track fit

KalTest Kalman fitter package

- KalTest
  - Kalman Fitting library (Keisuke Fuji et al)
    - recently migrated code base to SVN
    - added cmake build scripts
- KalDet
  - detector description (geometry and material) for KalTest
    - migrated to SVN
    - currently writing the geometry build up from GEAR
- to be released in next iLCSoft release v01-10
- both packages will be used by LCTPC (MarlinTPC) and ILD / iLCSoft
  - -> try to share as much common code as possible, i.e. is reasonable given the slightly different requirements for testbeam and global detector optimization

KalTest library

- based on ROOT
  - TGeo, TMath, TObjArray
- structured in sub-libraries
  - geomlib - geometry
  - kallib - Kalman filter
  - kaltracklib - Kalman tracking
  - utils - utilities
- built into one libKalTest.so
- users need to define their detector classes (KalDet):
  - TVMeasLayer
    - meas. layer, coordinate to track state transform. ...
  - TVDetector
    - position of meas. layer and material properties

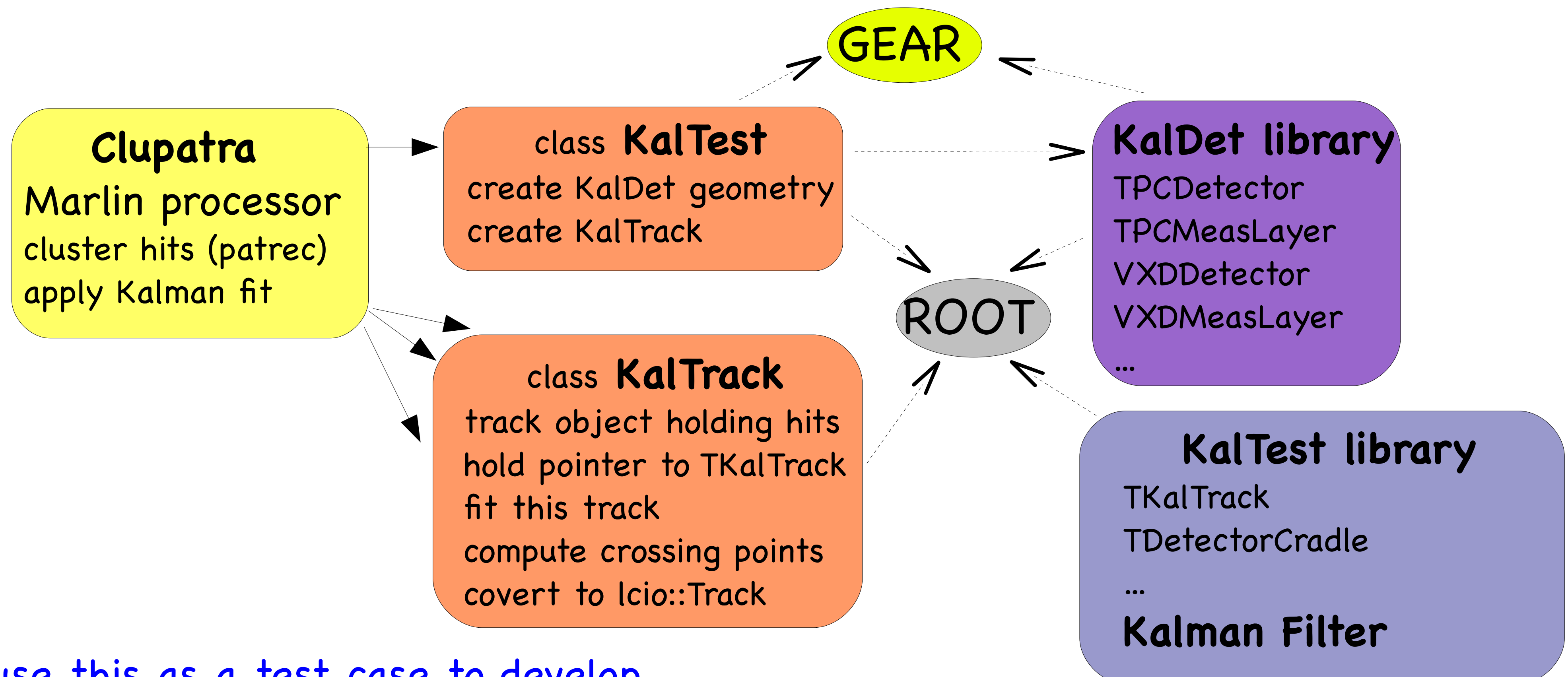


- track parameters correspond to LCIO, except:
  - d0_lcio = - drho_kaltest
  - omega_lcio = a(cB) * kappa_kaltest
- and different units:
  - KalTest: cm, KGauss, GeV
  - LCIO: mm, Tesla, GeV

plan to adapt Kaltest units to LCIO

interface to KalTest

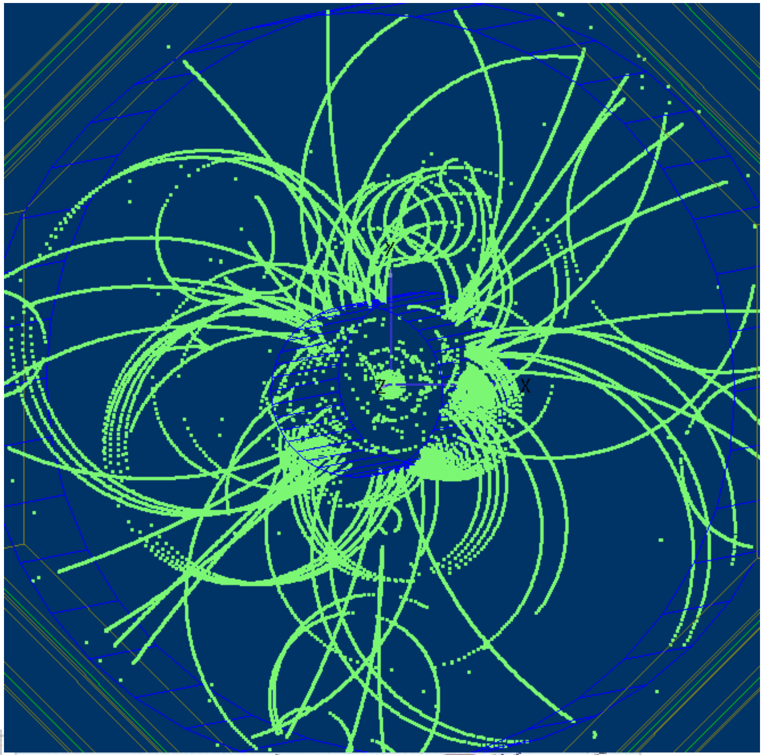
- need interface to KalTest fitter
- would like to have loose coupling between patrec and fitting
- need several iterations between patrec and fitting
  - LCIO::Track class not optimal for that (not designed to be)

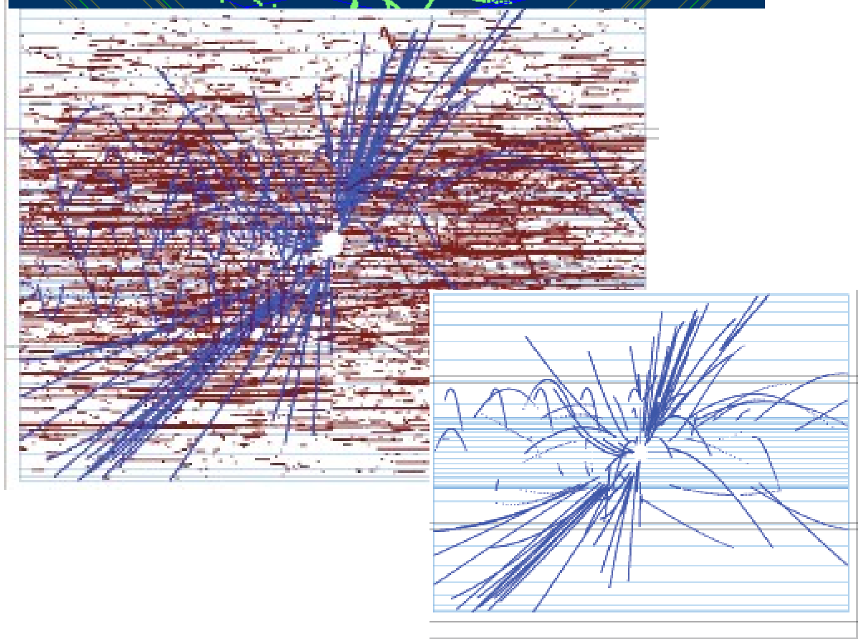


use this as a test case to develop
an API that can be used to write patrec
transparent to underlying track fitting code ...

package dependency:
using class:

TPC Pattern recognition





- patrec in a TPC should be rather easy
  - tracks immediately visible
  - "could be done by a kid with crayons"
- ILD TPC has a huge number of voxels >200 hits on many tracks
- classic triplet search and combinatorial Kalman filter probably overkill (CPU & coding intensive)
- mean distance between hits on track is mostly much smaller than distance between tracks
- => could use NN-CLustering
- NB: micro curlers from pair bg should be removed beforehand

Generic NN-clustering algorithm

```
template <class In, class Out, class Pred >
void cluster( In first, In last, Out result, Pred* pred ) {

    typedef typename In::value_type GenericHitPtr ;
    typedef typename Pred::hit_type HitType ;

    typedef std::vector< GenericCluster<HitType >* >  ClusterList ;

    ClusterList tmp ;
    tmp.reserve( 256 ) ;

    while( first != last ) {

        for( In other = first+1 ; other != last  ; other ++ ) {

            if( pred->mergeHits( (*first) , (*other) ) ) {

                if( (*first)->second == 0 && (*other)->second == 0 ) {  // no cluster exists

                    GenericCluster<HitType >* cl = new GenericCluster<HitType >( (*first) ) ;

                    cl->addHit( (*other) ) ;

                    tmp.push_back( cl ) ;

                }
                else if( (*first)->second != 0 && (*other)->second != 0 ) { // two clusters

                    (*first)->second->mergeClusters( (*other)->second ) ;

                } else {  // one cluster exists

                    if( (*first)->second != 0 ) {
                        (*first)->second->addHit( (*other)  ) ;
                    } else {
                        (*other)->second->addHit( (*first)  ) ;
                    }
                }
            } // dCut
        }
        ++first ;
    }
    // remove empty clusters
```

simplest *nearest neighbor clustering:*

- loop over all hit pairs
- merge hits into one cluster if *d(h1, h2) < cut*
- d() could be 3D-distance
  - typically more complicated

- generic NN-Clustering
  - template based
  - works for any C++ class
  - uses std::lists for clusters
- main classes
  - GenericHit< MyClass >
  - GenericCluster< MyClass >
- use predicate class for cut
- released in MarlinUtil as NNClusters.h
- improved and extended for TPC patrec…

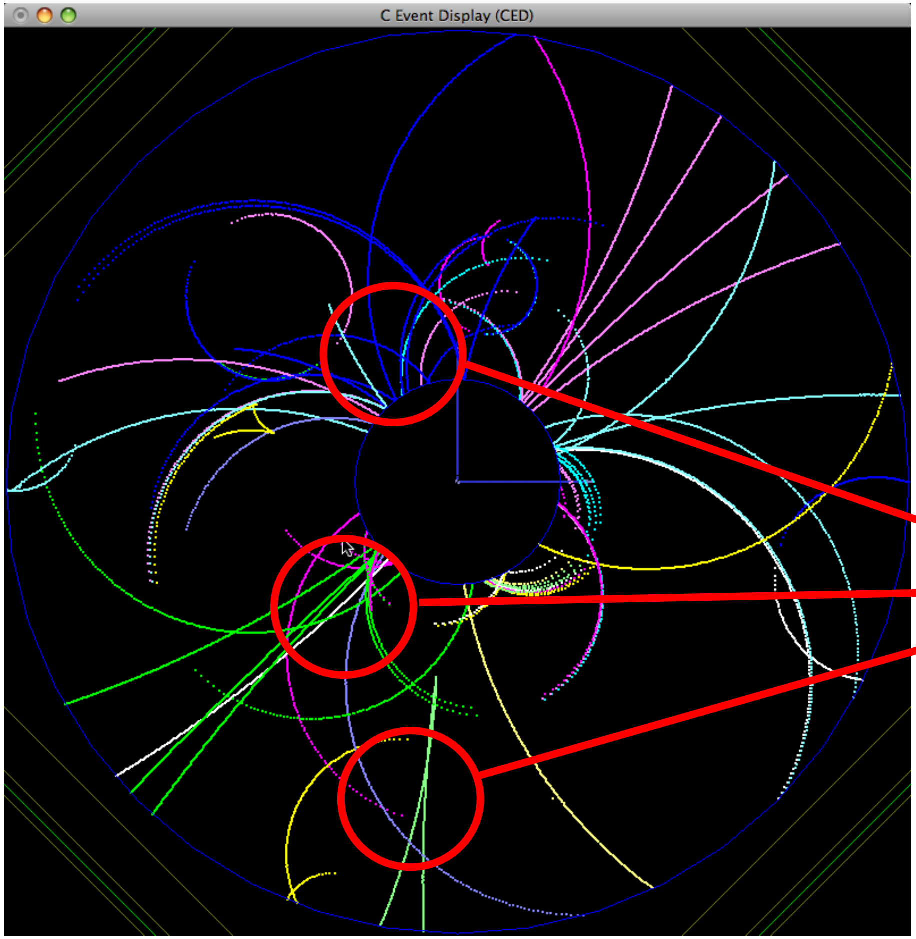
Application to TPC patrec I

- use GenericHit<lcio::TrackerHit>
- use simple euclidian distance
  - d = sqrt(dx^2 + dy^2 + dz^2) < C ; C = 40mm
- use z-index + sliding window to speed up processing

```
inline bool mergeHits( GenericHit<HitClass>* h0,
                       GenericHit<HitClass>* h1) {

  const PosType* pos0 =  h0->first->getPosition() ;
  const PosType* pos1 =  h1->first->getPosition() ;

  return
    ( pos0[0] - pos1[0] ) * ( pos0[0] - pos1[0] ) +
    ( pos0[1] - pos1[1] ) * ( pos0[1] - pos1[1] ) +
    ( pos0[2] - pos1[2] ) * ( pos0[2] - pos1[2] )
    < _dCutSquared ;
}
```



example:
- ttbar event @ 500 GeV

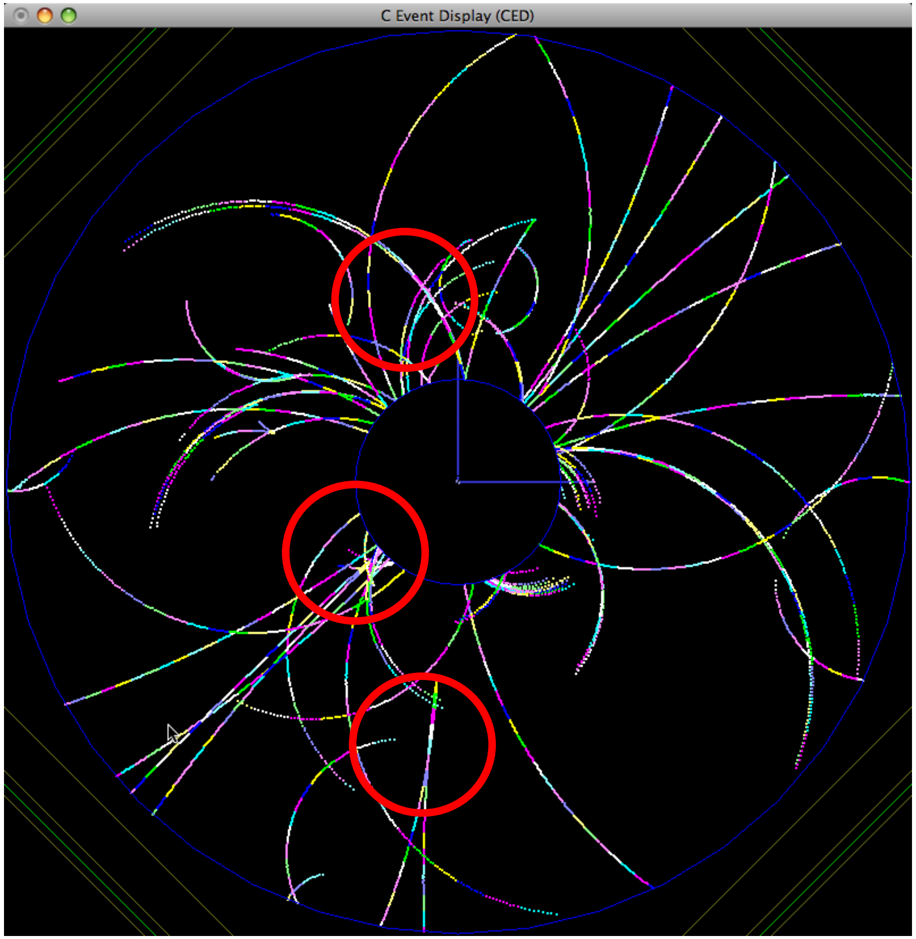
obvious issue:
- close by tracks are merged into one cluster

Application to TPC patrec II

fix falsely merged tracks:

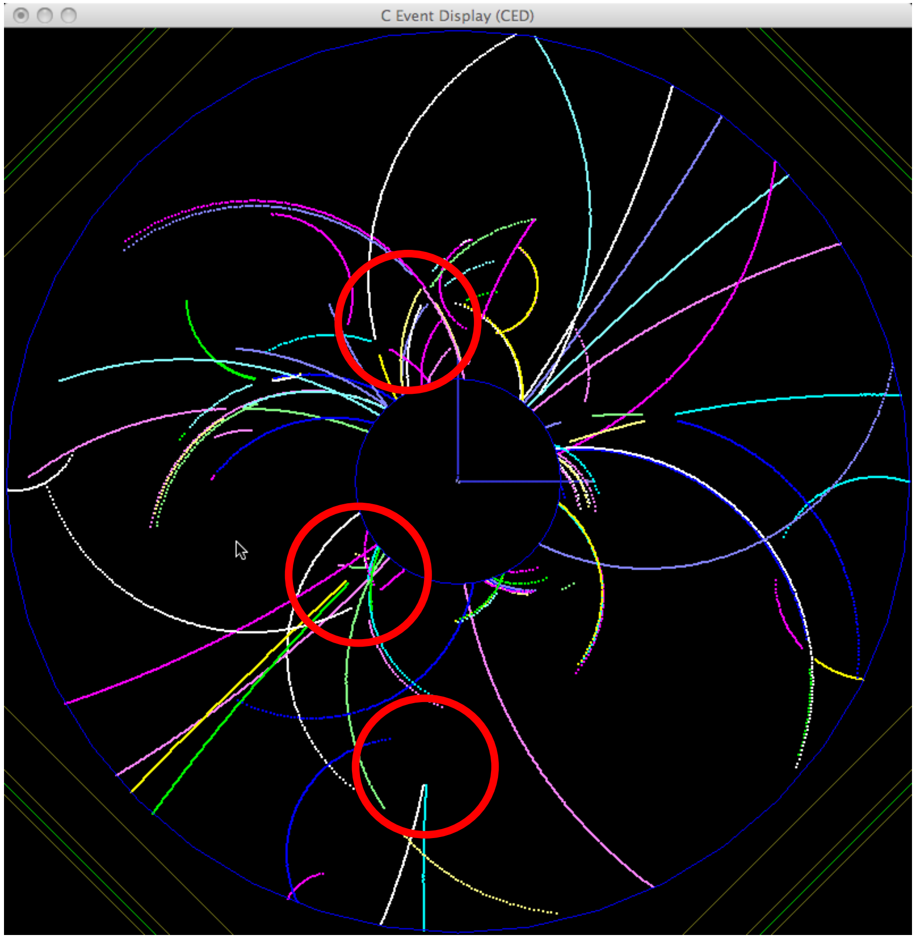
- identify clusters with duplicate pad rows
- recluster in pad row ranges (10 pad rows)
- remove sub-clusters with hits in duplicate pad rows
- recluster all hits

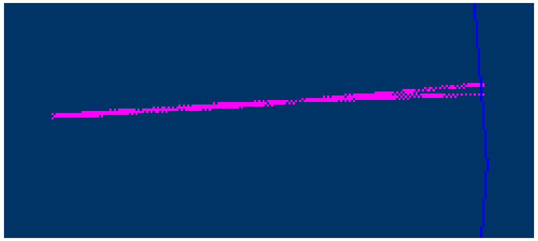
- iterate with shifted pad row ranges (5 pad rows)



Application to TPC patrec III

- issues – again obvious:
- some hits lost due to duplicate pad rows
- loose nearby tracks - not separated at all along full length

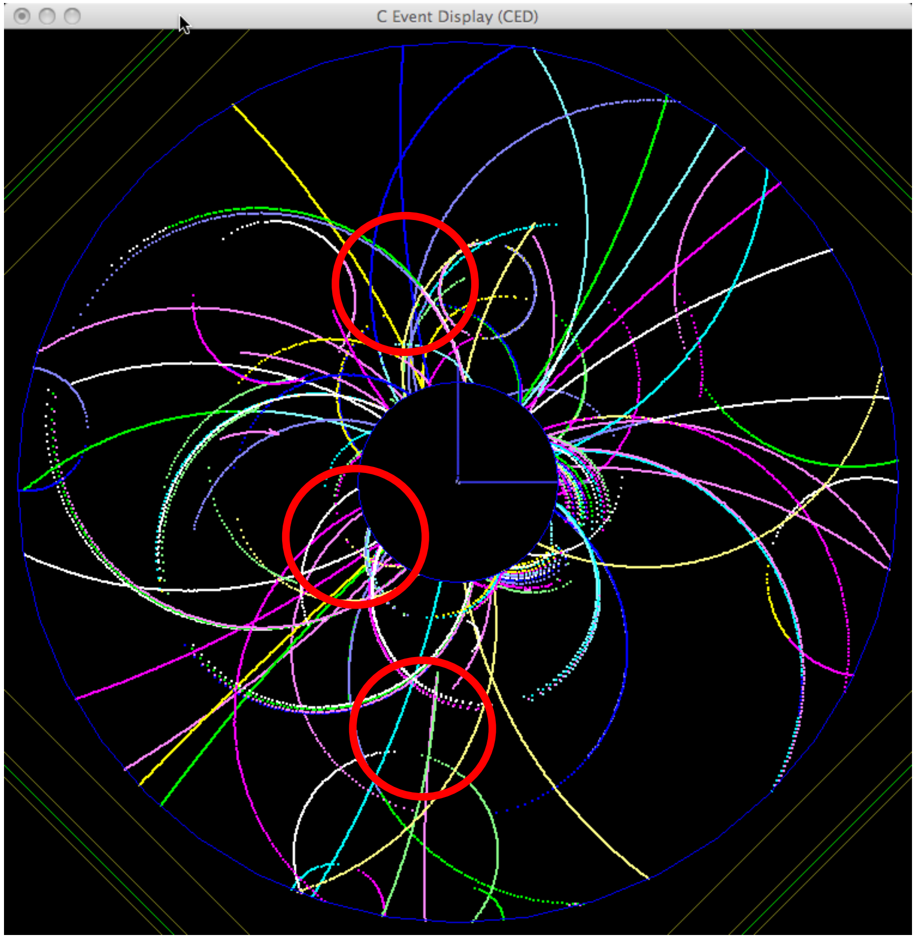




example: V0 close to outer field cage

use KalTest to assign left over hits

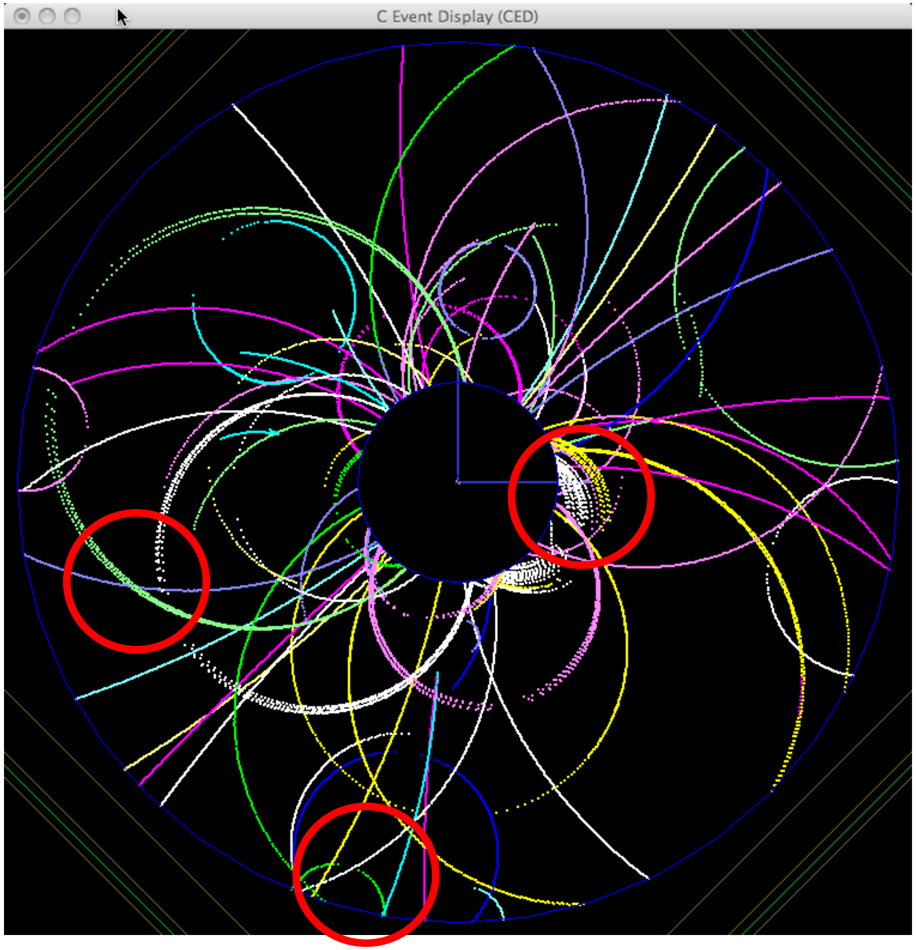
- fit all track segments with KalTest
- compute all crossing points of helix with layers – inwards and outwards of cluster segment
- assign leftover hits to best matching helix
- refit new track segments



- to do
- merge track segments
  - based on track state
  - take error into account

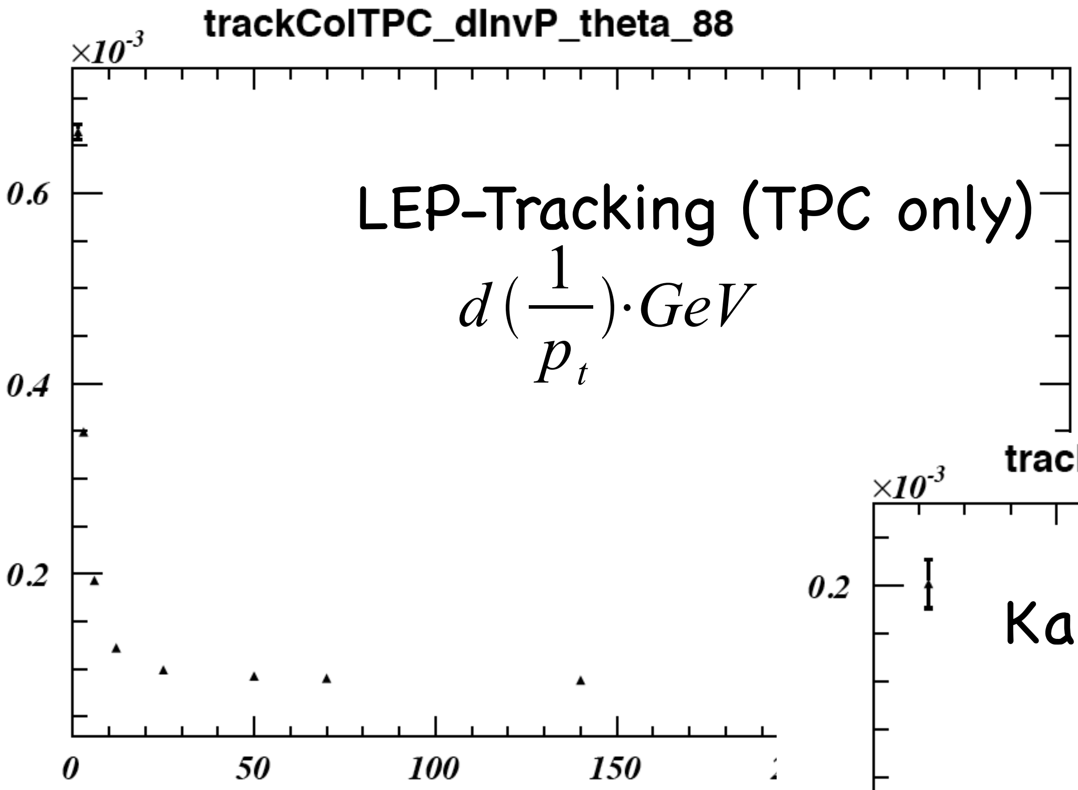
merging track segments

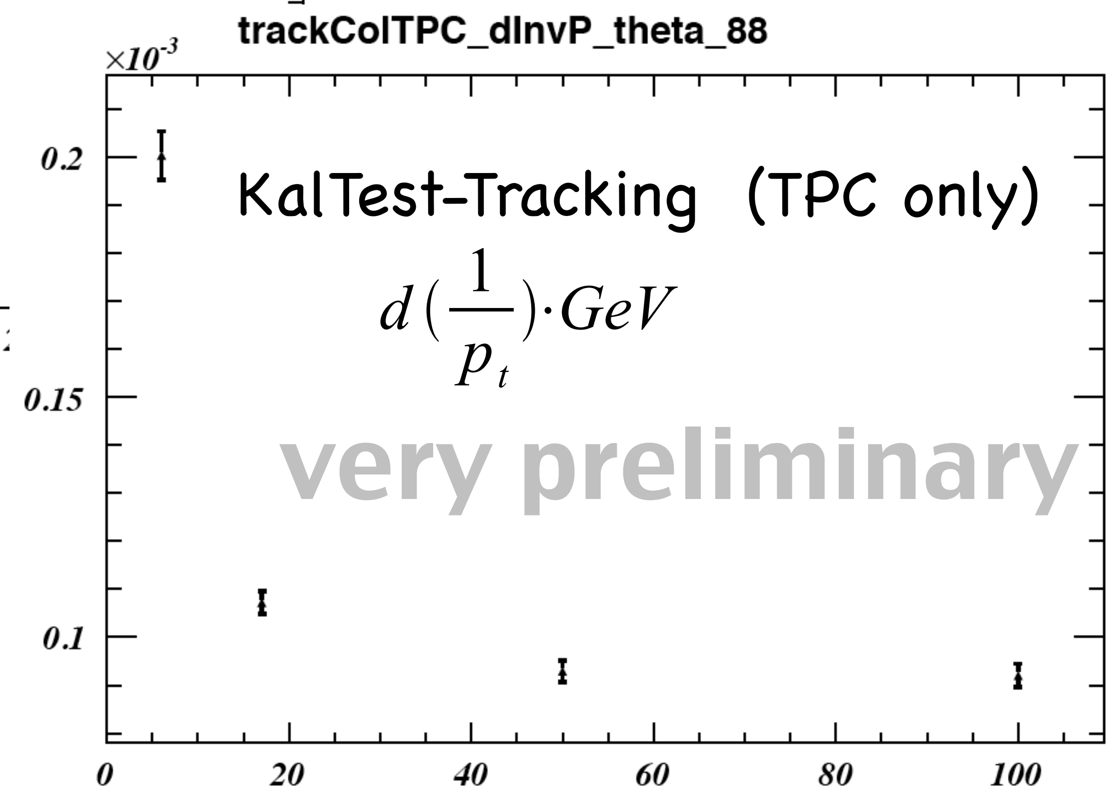
- for now merge tracks in r-phi: radius and center only
  - (use NNCluster to 'cluster' track segments)
- -> can merge curlers (previously lost in f77-TPC tracking)
- could improve PFA track cluster matching for low pt particles going to endcaps -> to be studied



- to do
- study track segment merging quantitatively
- possibly have a better defined merging criterion
- e.g:
- Chi2 between track states:
  - chi2 = ts0 * 1/C * ts1

single particles - momentum resolution

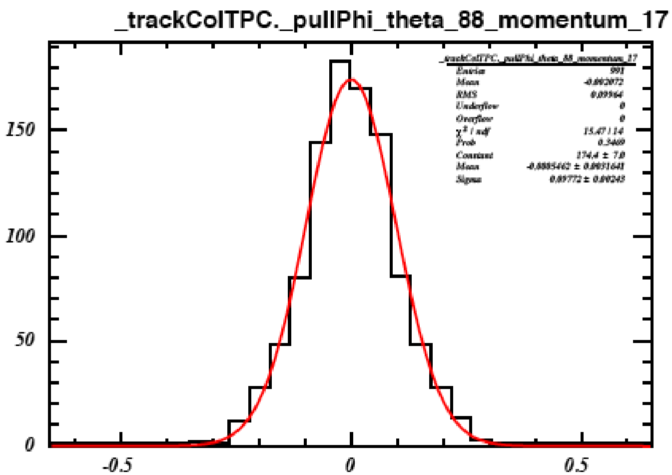


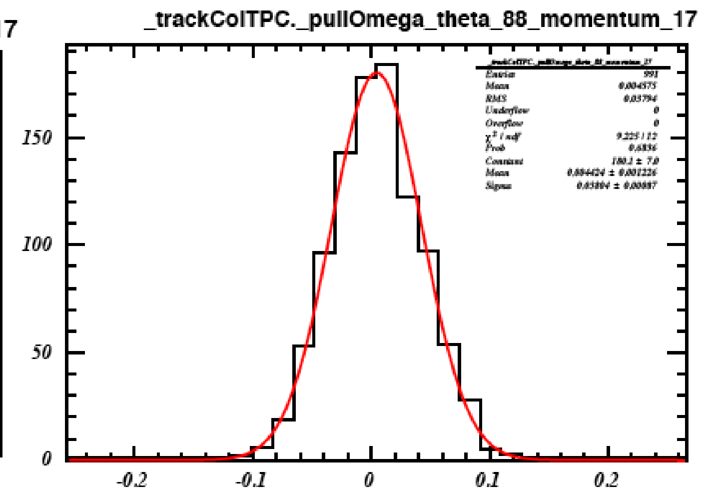


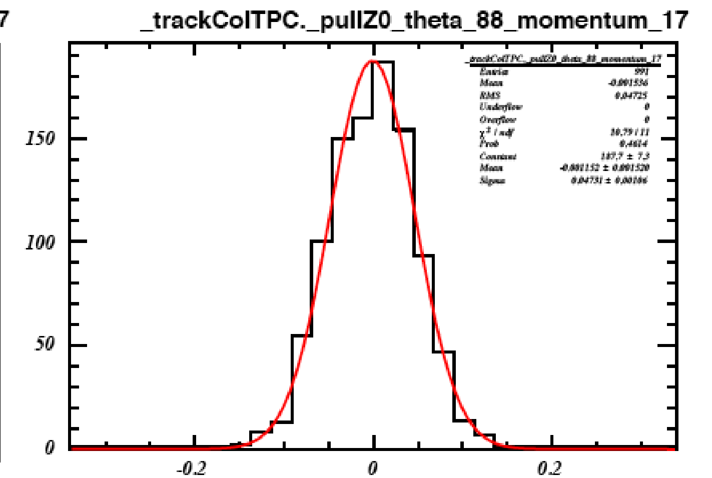
- single mu+
- theta=88 deg
- p = 6, 17, 50, 100 GeV

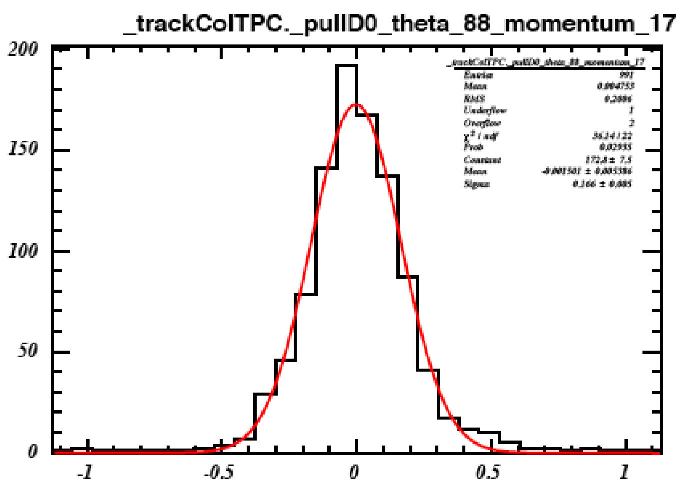
plots: S.Aplin

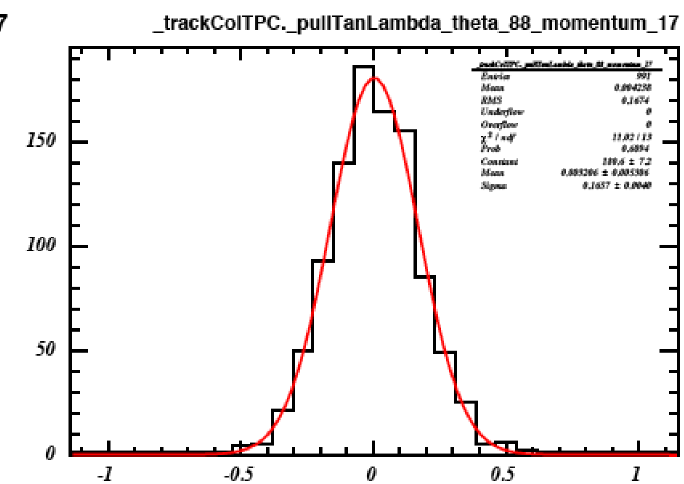
single particles – pull distributions











pull=(meas−true)/Error_meas

- single mu+
- theta = 88 deg
- p = 17 GeV

plots: S.Aplin

- measurment errors are wrong - need to understand this !
  - hit errors given to KalTest
  - conversion of track parameters - covariance matrix (unit conversions)
  - make sure material description is correct
  - ...

work in progress ...

Summary & Outlook

- started to adapt KalTest Kalman fitter to iLCSoft
  - to be used by LCTPC and ILD
- developed TPC pattern recognition based on NN-clustering
- interfaced loosely to KalTest
  - used for fitting and extrapolation of track segments
- first look at fitting single particles:
  - fits work in principle - issues in pulls need to be addressed
- Outlook
  - debug and check interface to KalTest:
    - material description, hit errors, unit conversions,...
  - complete merging of track segments
  - systematic studies on different physics channels
  - parameter tuning
  - add SIT, VXD,...