Reco Changes

Bruce Baller
July 14, 2015

Outline

- ClusterFinder/LineCluster_module - update
- RecoAlg/ClusterCrawlerAlg - update
- TrackFinder/CCTrackMaker_module – major update
- RecoAlg/TrackTrajectoryAlg - update
- RecoAlg/VertexFitAlg – new

- Much of this information is for reference and will not be described in detail
- Highlights for this meeting in red

LineCluster_module

- Calls ClusterCrawlerAlg
- Produces

```
LineCluster::LineCluster(fhicl::ParameterSet const& pset) {
  reconfigure(pset);

  // let HitCollectionAssociator declare that we are going to produce
  // hits and associations with wires and raw digits
  // (with no particular product label)
  recob::HitCollectionAssociator::declare_products(*this);

  produces< std::vector<recob::Cluster> >();
  produces< std::vector<recob::Vertex> >();
  produces< std::vector<recob::EndPoint2D> >();     ← "New"
  produces< art::Assns<recob::Cluster, recob::Hit> >();
  produces< art::Assns<recob::Cluster, recob::Vertex, unsigned short> >();
  produces< art::Assns<recob::Cluster, recob::EndPoint2D, unsigned short> >();  ← "New"
} // LineCluster::LineCluster()
```

- Interface is unchanged

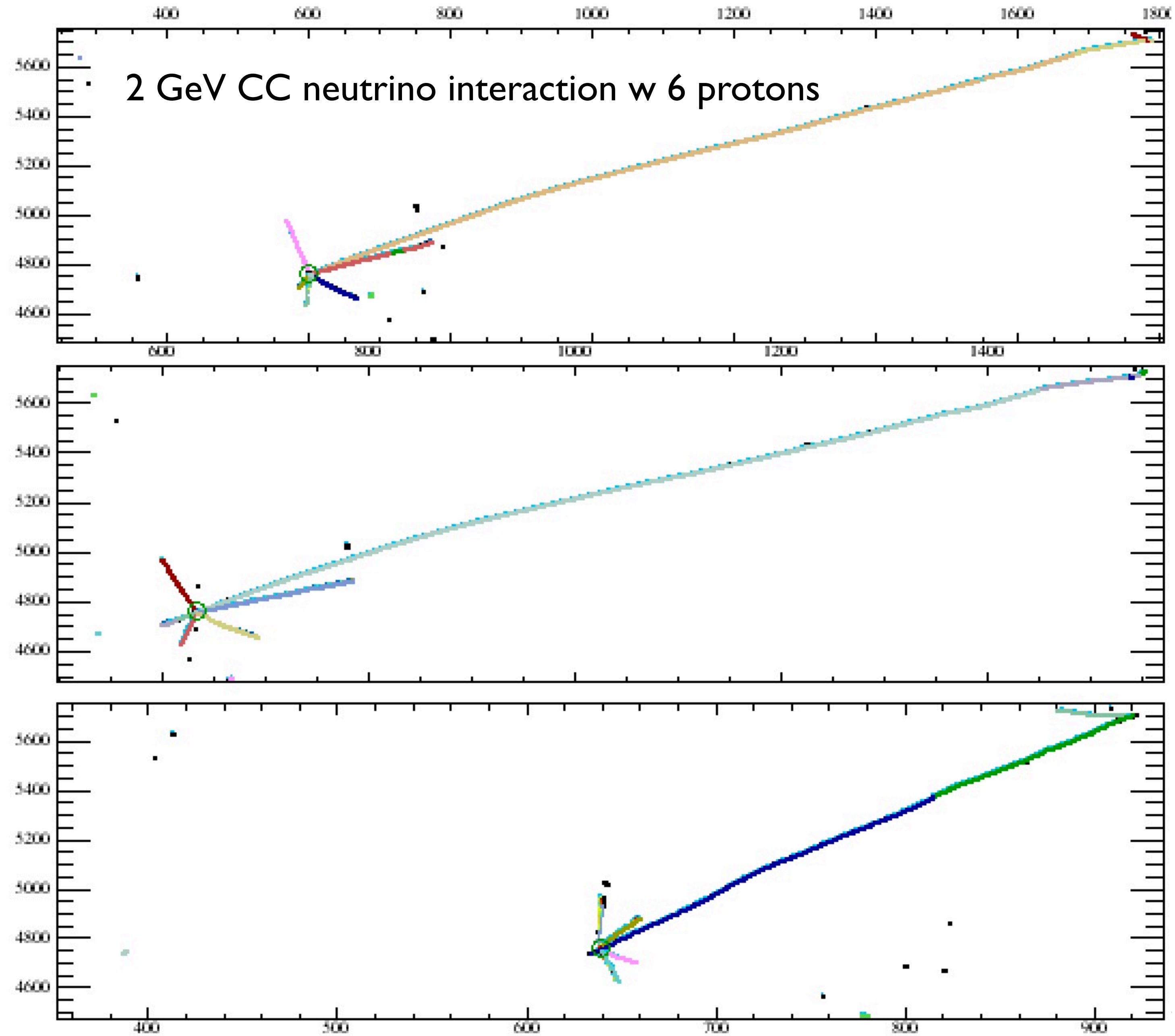
ClusterCrawlerAlg Updates

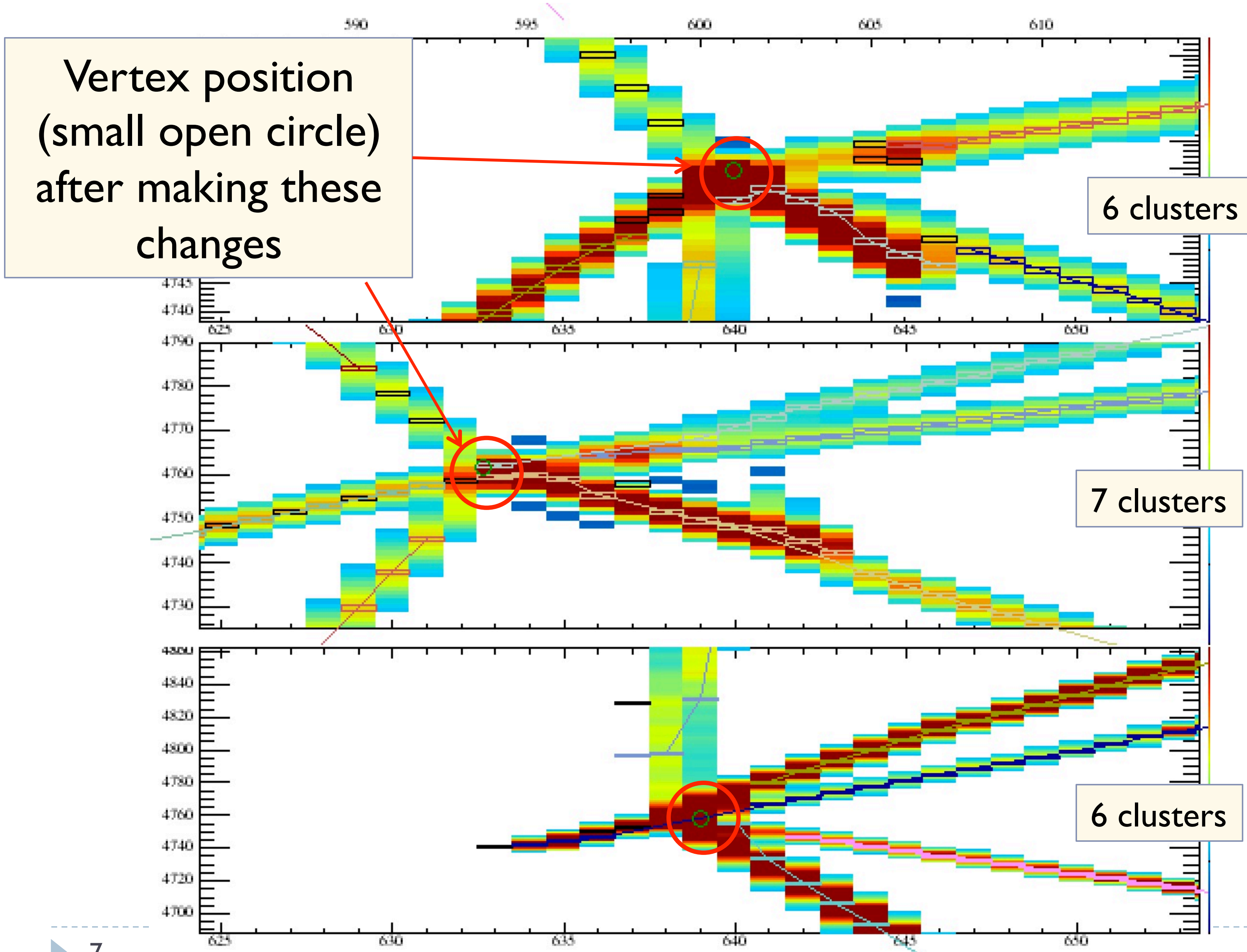
- Vertex finding & fitting
- Hammer clusters new
- ChkSignal revision
- MergeOverlap new – merges overlapping clusters

ClusterCrawlerAlg - *2D Vertex Finding*

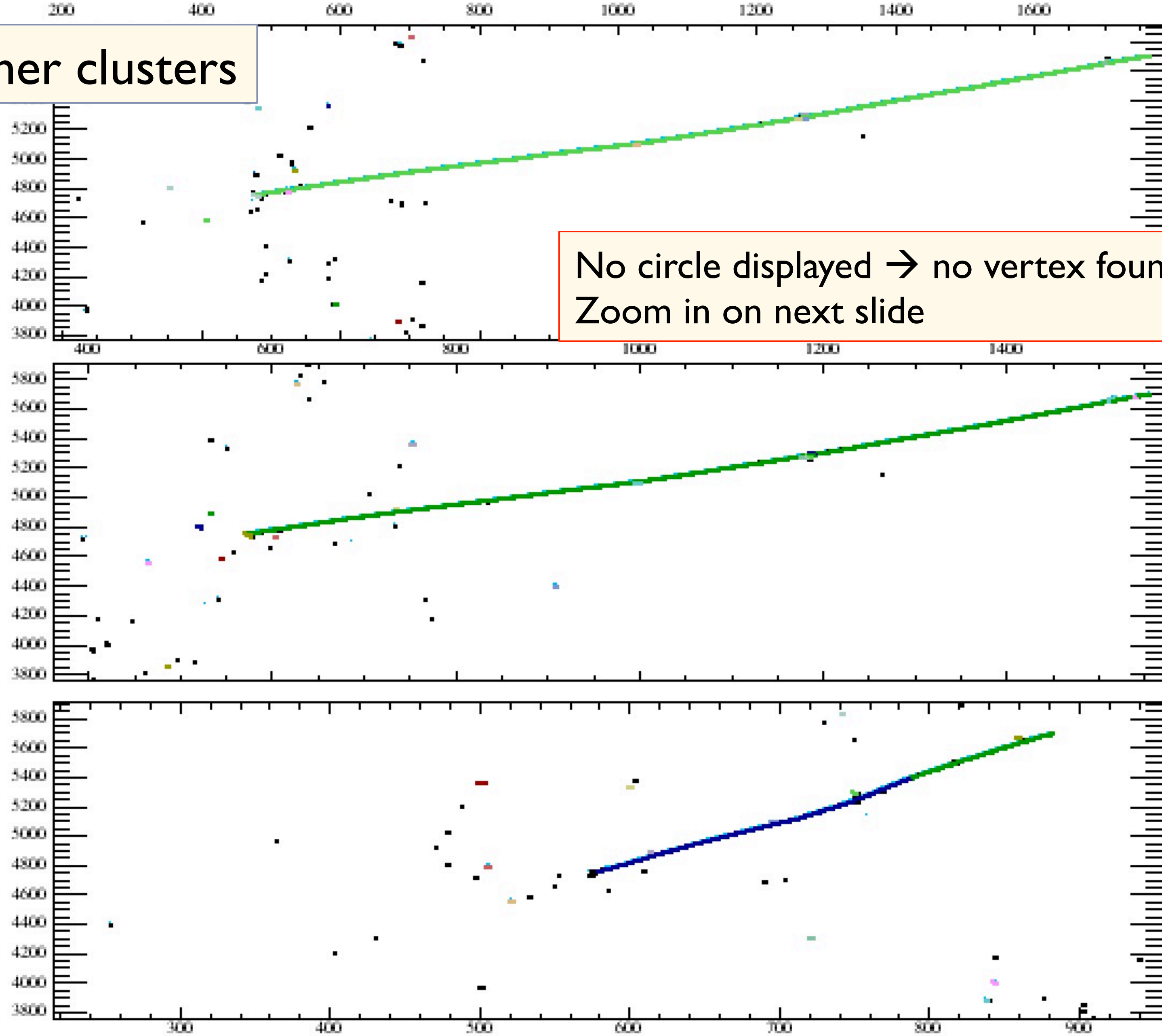
- Algorithm (old)
  - Double loop over cluster pairs that have no Vtx assignment
  - Calculate (v_{wire}, v_{tick}) of the intersection point
  - Ensure that there is hit charge on all wires between (v_{wire}, v_{tick}) and the Begin/End (wire, tick) of both clusters (using ChkSignal)
  - Compare (v_{wire}, v_{tick}) with set of existing vertices and merge them if δWire < 4 and δTick < 25
- Problem
  - Poor cluster reconstruction near the vertex can result in nearby vertices that fail the cuts particularly for large angle clusters
- Solution
  - Modify FitVtx routine to save the vertex position error
  - Replace the hard-coded δWire and δTick cuts with chisq cuts

- Side note: recob::Vertex has no error on the (x,y,z) position

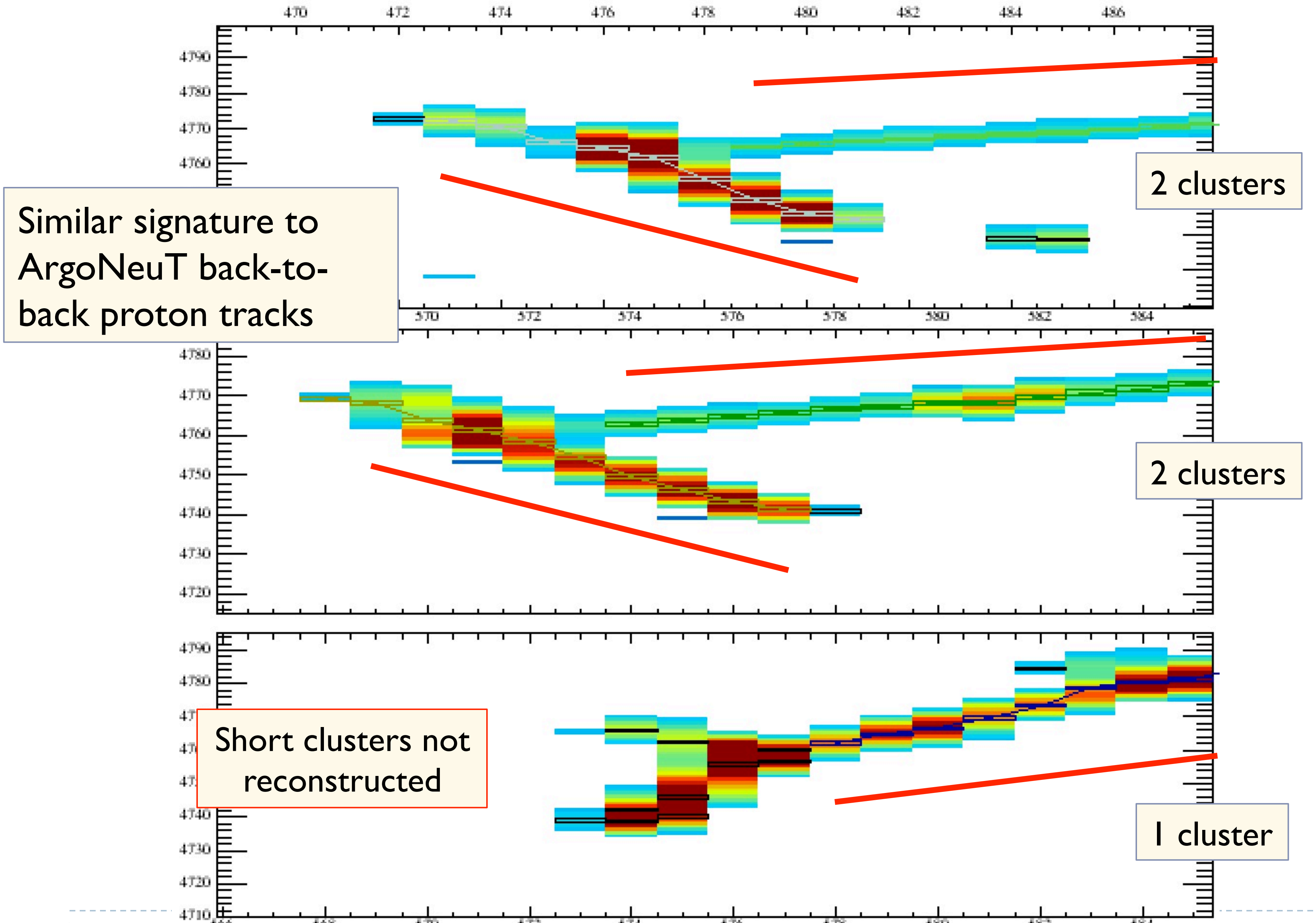

2 GeV CC neutrino interaction w 6 protons
400 600 800 1000 1200 1400 1600 1800
5600 5400 5200 5000 4800 4600
600 800 1000 1200 1400
400 500 600 700 800 900


Vertex position (small open circle) after making these changes
6 clusters
7 clusters
6 clusters

Hammer clusters



No circle displayed → no vertex found
Zoom in on next slide

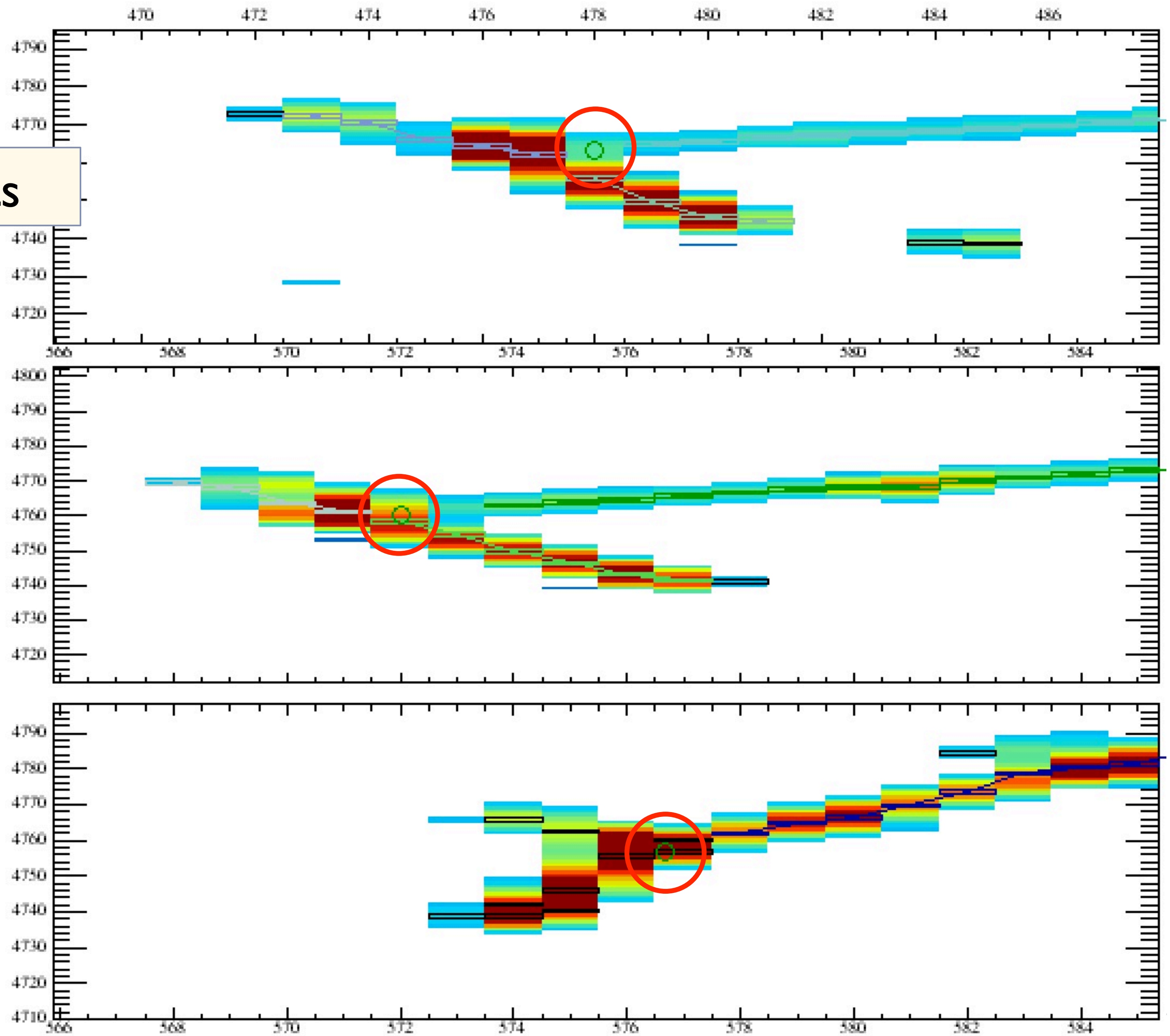


Similar signature to ArgoNeuT back-to-back proton tracks
2 clusters
2 clusters
Short clusters not reconstructed
1 cluster

ClusterCrawlerAlg
New routine - FindHammerClusters

- In each plane, look for:
  - One long (>20 hits) cluster whose End is near a short (< 20 hits) cluster
  - An (v_{wire}, v_{tick}) intersection point on the short cluster where v_{wire} < End wire of the long cluster
  - Calculate the X position of the intersection point
  - Store in a temporary struct
- Match in 3D
  - Create an "incomplete" 3D vertex (i.e. 2/3 planes match) if:
    - The X position of hammer clusters are < fVertex3Dcut
    - X, Y, Z position of the matched 2D vertices is within the TPC
    - Split the short clusters in the two matching planes
    - Create 2D vertices in the two matching planes and assign the short and long clusters to them
- Use existing Vtx3ClusterSplit routine to try to "complete" the 3D vertex in the third plane

Results

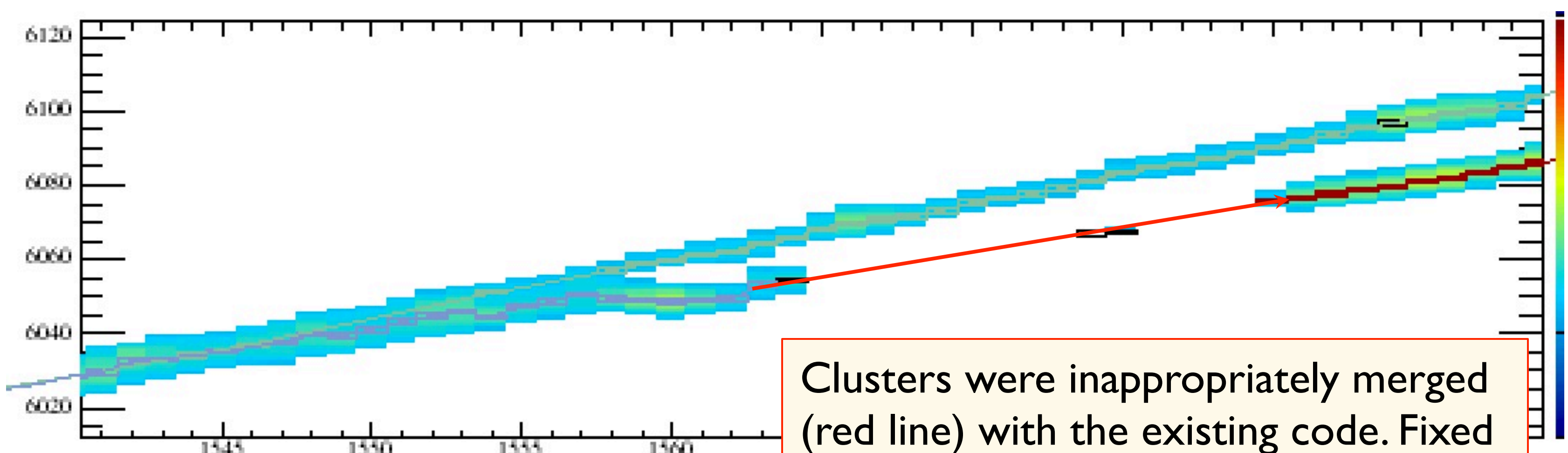


470
472
474
476
478
480
482
484
486
4790
4780
4770
4740
4730
4720
566
568
570
572
574
576
578
580
582
584
4800
4790
4780
4770
4760
4750
4740
4730
4720
4790
4780
4770
4760
4750
4740
4730
4720
4710
566
568
570
572
574
576
578
580
582
584

ClusterCrawlerAlg

ChkSignal reminder

- Returns true if there is a "wire signal" between ($wire_1$, $tick_1$) and ($wire_2$, $tick_2$) (red line in the figure)
  - Original (pre-LArSoft) version checked wire ADC values
    - Not available when wire signals were dropped from the event
  - Converted to check proximity of hits (PeakTimeMinusRMS, PeakTimePlusRMS) to the expected tick → OLD



Clusters were inappropriately merged (red line) with the existing code. Fixed with the changes described →

ClusterCrawlerAlg
ChkSignal revision

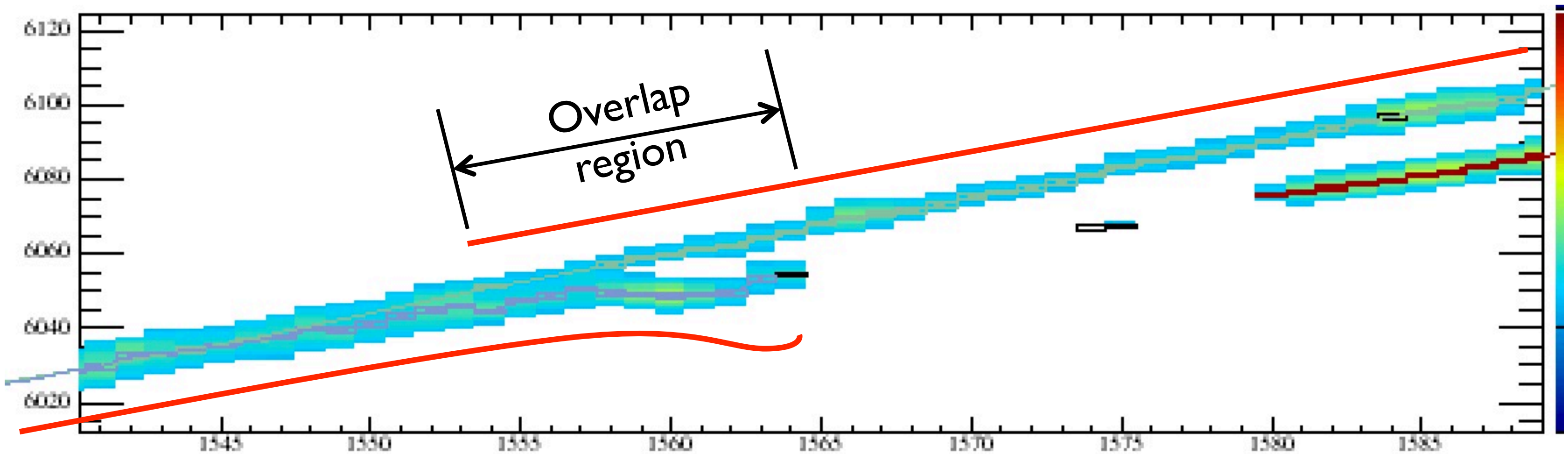
- Correct method is to require that the wire signal amplitude on each wire along the line between ($wire_1$, $tick_1$) - ($wire_2$, $tick_2$) > expected wire signal (t)
- Requires calculating the hit amplitude A(t) = PeakAmplitude * exp (-0.5 * (t - PeakTime)^2 / RMS)
  - Or use a Gaussian histogram to speed things up

```
// Gaussian amplitude in bins of size tBin
const float tBin = 0.15;
const float gausAmp[20] = {1, 0.99, 0.96, 0.90, 0.84, 0.75, 0.67, 0.58, 0.49, 0.40, 0.32, 0.26, 0.20,
    0.15, 0.11, 0.08, 0.06, 0.04, 0.03, 0.02};
```

- New fcl parameter MinAmp (= 10 for uB) for the minimum wire signal amplitude

MergeOverlap

- Poorly reconstructed (broken) clusters may overlap each other – especially cosmic rays



- Merge clusters using fit information just outside of the overlap region to select hits within the overlap region
  - Also merge hits in the overlap region if the charge of merged hits is consistent with charge of hits outside the overlap region

ClusterCrawlerAlg

Code Development Output Improvements

```
****** 3D vertices ******************__2DVtx_Indx__*******
Vtx  Cstat  TPC     X       Y       Z  pln0 pln1 pln2  Wire
  0      0    0 -253.1    95.3   323.8    0    9   -1 1078    Incomplete match
  1      0    0  140.5     2.5   187.6    2   12   -1  624    Incomplete match
  2      0    0  140.6    -9.5   166.8    2   23   18   -1    Matched in 3 planes
  3      0    0  140.7    29.8   140.2    6   12   -1  466    Incomplete match
  4      0    0  141.5    33.4   146.4    6   -1   16  673    Incomplete match
  5      0    0  140.9    45.2   166.8    6   -1   18  741    Incomplete match
  6      0    0  143.2    59.4   157.8    7   -1   17  767    Incomplete match
  7      0    0   29.5    89.1   827.1   22    8   14   -1    Matched in 3 planes
  8      0    0  151.9   -17.7   160.8   -1   10   19  657    Incomplete match
  9      0    0  143.7    37.0   157.8   24   11   17   -1    Matched in 3 planes
 10      0    0  140.4    14.5   166.8   25   12   18   -1    Matched in 3 planes
************ 2D vertices ************
Vtx   CTP    wire      error   tick      error  ChiDOF  weight  topo  cluster IDs
 14     2  2756.0 +/-    1.7 3574.9 +/-    7.6    0.0    41.0     1    326 342
 15     2  1067.0 +/-    0.5   16.4 +/-    1.0    0.0    41.0     1    322 332
 16     2   487.0 +/-    0.5 4972.9 +/-    1.0    0.0    15.0     2    404 413
 17     2   525.0 +/-    0.5 4993.9 +/-    1.1    0.0    19.0     2    336 383
 18     2   555.0 +/-    0.5 4957.8 +/-    1.0    0.0    13.0     3    333 379
 19     2   535.0 +/-    0.5 5100.9 +/-    1.0    0.0    14.0     2    334 384
 20     2  2619.0 +/-    1.0 6924.9 +/-    2.0    0.0    10.0     3    357 359
*************************************** Clusters *****************************************************
  ID CTP nht Stop  Proc  beg_W:T    bAng   bSlp bChg  bCN   end_W:T    eAng   eSlp eChg  eCN bVx  eVx aveRMS
 315  2  460   0      0 3455:4132   0.44   1.77  104  0.0  2984:3301   0.44   1.77  128  0.0  -99  -99  2.2
-316  2  116   0      0 3454:8753   0.07   0.26  134  0.1  3339:8722   0.07   0.28  143  0.0  -99  -99  2.3
-317  2  492   0      0 3378:5100   0.19   0.73  129  0.1  2879:4824   0.13   0.49   94  0.3  -99  -99  2.4
-318  2  232   0    300 2988:4423   0.90   4.66  182  0.0  2756:3573   0.89   4.60  135  0.4  -99  -99  2.5
 319  2  131   0      0 2720:7170   1.05   6.52  180  0.2  2586:6227   1.11   7.52  205  1.4  -99  -99  4.7
 320  2  143   0    300 2616:6456   1.17   8.87  137  0.7  2461:5154   1.17   8.83  411  0.0  -99  -99  4.8
 321  2  359   0      0 1423:6839  -0.58  -2.45  158  0.1  1065:7720  -0.59  -2.48   76  0.0  -99  -99  2.8
 322  2  223   0      0 1289:876    0.80   3.87   85  0.2  1067:15     0.80   3.88  126  1.0  -99   15  3.5
 323  2  145   0      0  145:7420  -0.37  -1.43  156  0.0     0:7627  -0.37  -1.44  178  0.2  -99  -99  2.3
```

→ recob:EndPoint2D's
→ recob::EndPoint2D's
→ recob::Vertex

Modified fcl file

```
standard_clustercrawleralg:
{
  NumPass:                 3  # number of passes through the hit list. 0 = no cluster reco
  MaxHitsFit: [ 100,   8, 4] # number of hits fitted to a line
  MinHits:    [ 50,   8, 3] # minimum size of a cluster
  NHitsAve:   [ 20,   8, 0] # number of hits to find the average charge and wid
                             # at the end of the cluster. NHitsAve should be 1 or 2
  ChgCut:    [ .8, .8, .8] # max fractional hit charge difference for adding hits
  ChiCut:     [4., 4., 4.] # stop adding hits to clusters if ChiCut is reached
  MaxWirSkip:   [25, 8, 2] # max number of wires to skip without adding a hit
  MinWirAfterSkip: [2, 2, 1] # min reqd number of consecutive wires with a hit after a skip
  KinkChiRat: [1.2, 1.2, 0.] # Max consecutive chisq increase for the last 3 hits on the cluster
                             # 0. = no kink check when following
  KinkAngCut: [0.4, 0.4, 0.4] # kink angle cut (radians) used to follow and merge
  DoMerge:   [false, true, true] # run cluster merging code?
  TimeDelta: [2., 3., 10.] # max time difference for cluster merging
  MergeChgCut: [0.8, 0.8, 0.8] # max charge ratio for cluster merging
  FindVertices: [true, true, true]  # make 2D vertices after clustering?
  LACrawl:     [true, true, true]  # crawl Large Angle clusters?
  LAClusAngleCut:        60 # Large cluster angle cut (0 < 90 degrees). <0 to turn off
  LAClusMaxHitsFit:      10 #
  MinHitFrac:           0.6 # Drop clusters having < (#hits/#wires)
  MinAmp:                10 # Min hit signal amplitude for merging
  ChgNearWindow:         40 # #of ticks for summing charge near a cluster
  ChgNearCut:           1.5 # Cluster end is shower-like if (nearby chg)/(cls chg)> cut
  HitMergeChiCut:       1.5 # Merge cluster hit-multiplets if the separation chisq
                            # is < cut. Set < 0 for no merging
  MergeOverlapAngCut:   0.1 # Set <= 0 to turn off overlapping cluster merging
  ChkClusterDS:        true # Check reconstruction at DS end of clusters?
  VtxClusterSplit:     true # Split clusters that cross vertices
  FindStarVertices:    true # Find vertices with a star topology
  HitErrFac:            0.4 # hit time error for fitting = fHitErrFac * (hit width)
  AllowNoHitWire:         1 # Allow skipping N wires w no hits (if poor purity)
  Vertex2DCut:           10 # Max equiv dTick cut for attaching a cluster to a vtx
  Vertex3DCut:            3 # 2D vtx -> 3D vtx matching cut (chisq)
  HammerCluster:       true # look for hammer type clusters
  DebugPlane:            -1  # print info only in this plane
  DebugWire:              0  # set to the Begin Wire and Hit of a cluster to print
  DebugHit:               0  # out detailed information while crawling
}
```

New (MergeOverlapAngCut)

Old hard cut (cm) (Vertex3DCut)

New (HammerCluster)

CCTrackMaker_module

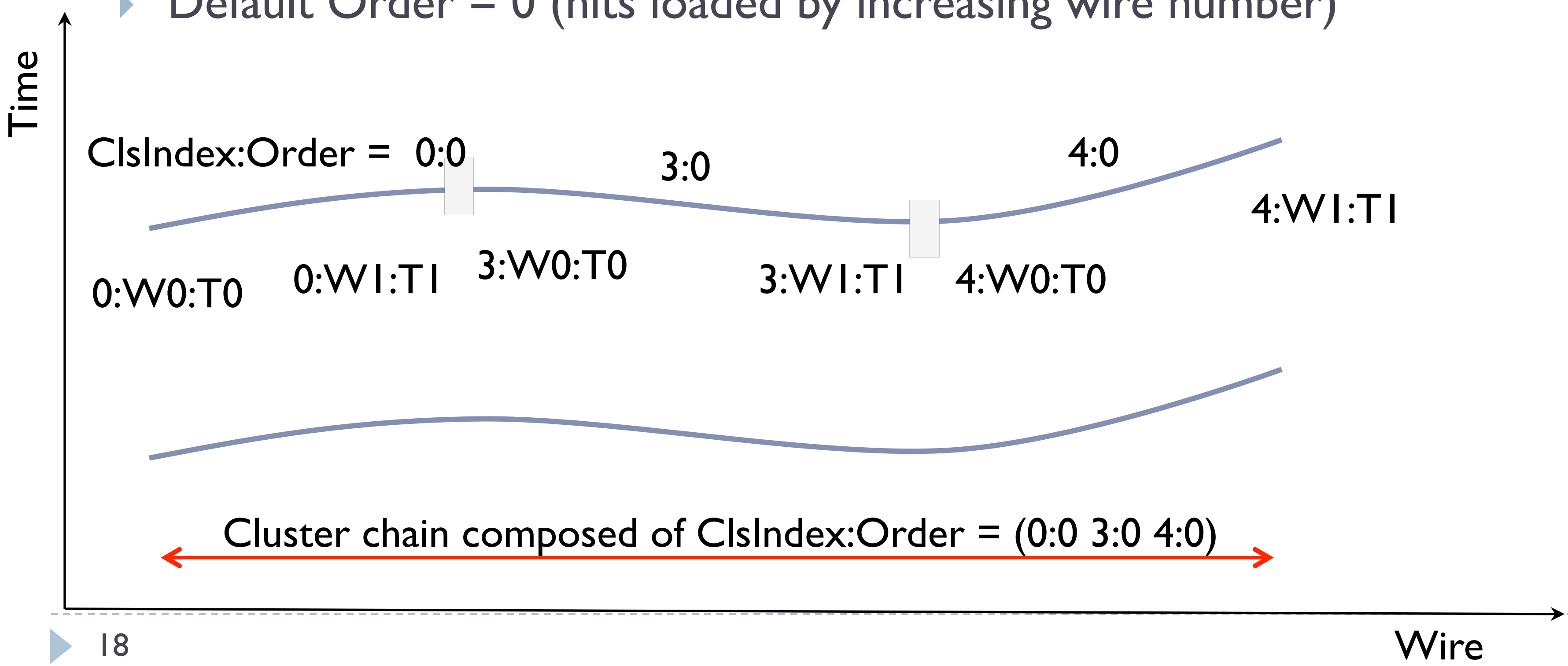
Algorithm

- Associate ends of broken clusters
  - Simplified scheme using "cluster chains" → New
- Two 3D cluster matching routines
  - VtxMatch uses clusters associated with 3D vertices found by ClusterCrawler
    - Match clusters that start(end) at the same vertex
  - PlnMatch matches clusters using cluster end point information
    - Several passes – long clusters, short clusters, etc → New
  - AngMatch matches clusters preferentially by angle New
- Both use FillEndMatch which finds a match error for 2 or 3 clusters at the "match end" (e.g. a common vertex or similar X) and the "other end" of the clusters
    - Wire[end], X[end], Angle[end] where end = 0 (US), 1 (DS)
    - Matching σ from fcl file: XMatchErr, AngleMatchErr
    - Match error ~ chgAsym * sqrt($\delta X^2/\sigma_X{}^2 + \delta A^2/\sigma_X{}^2 + \delta W^2/\sigma_W{}^2$)
      - chgAsym = 1 + (BigChg – SmallChg) / (BigChg + SmallChg)
- Put results into a vector of match structs

CCTrackMaker_module

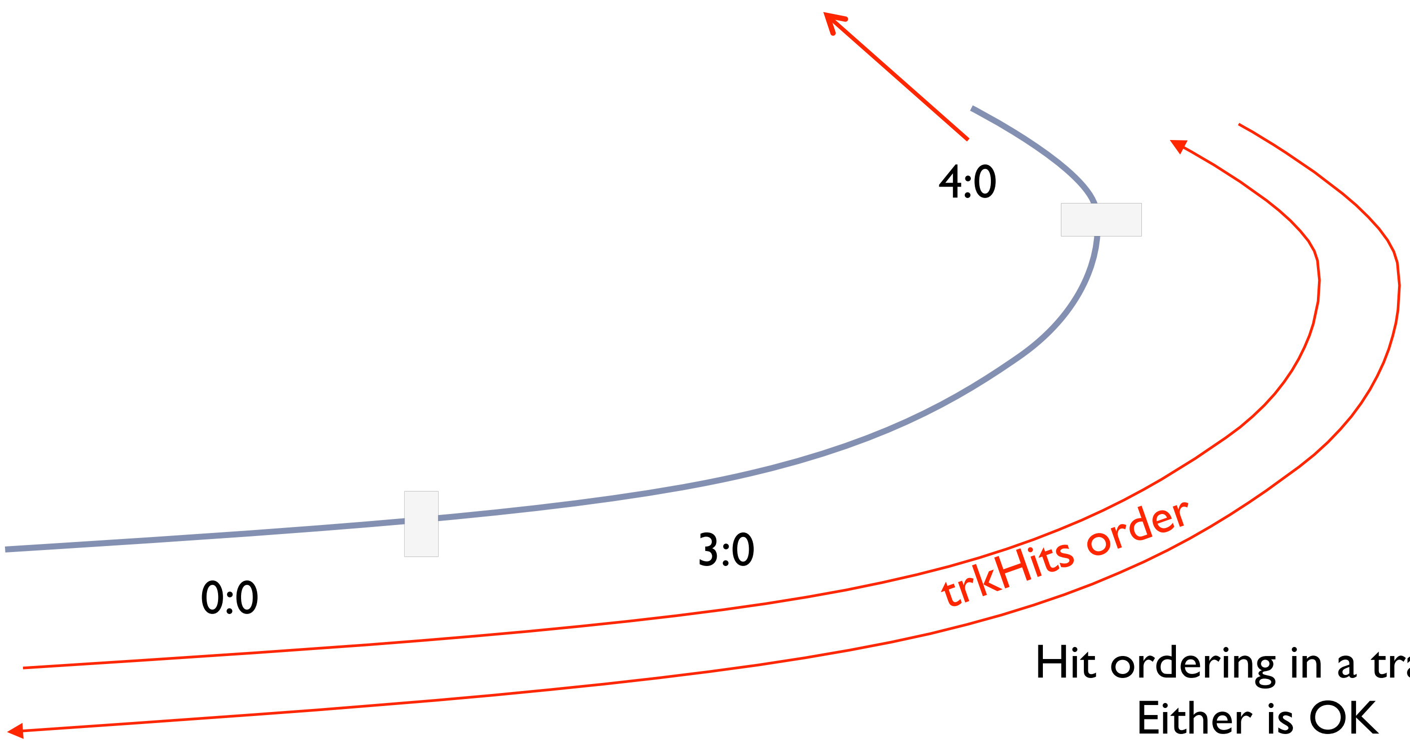
MakeClusterChains merges Broken Clusters

- ClusterCrawler cluster hits are naturally ordered by increasing wire number
  - Cluster "End" = 0 (US end, low wire num), 1 (DS End high wire num)
- Define a cluster Order for inserting in a trkHits vector
  - Consistent hit ordering between planes
  - Default Order = 0 (hits loaded by increasing wire number)

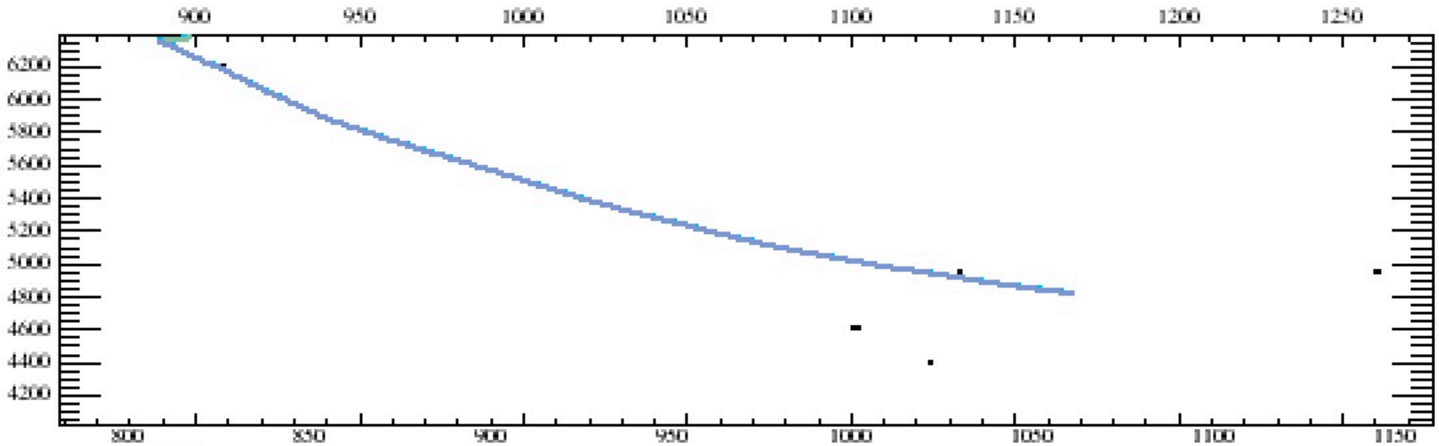


CCTrackMaker_module
Merging Wandering Clusters

- Cluster matching between planes loops over both ends of all clusters so the sequential ordering of clusters in the chain is irrelevant <u>but the hit order is</u>
  - Need ClsIndex:Order = (0:0 3:0 4:1) or (4:0 3:1 0:1)



Hit ordering in a track.
Either is OK
Until you match clusters between planes

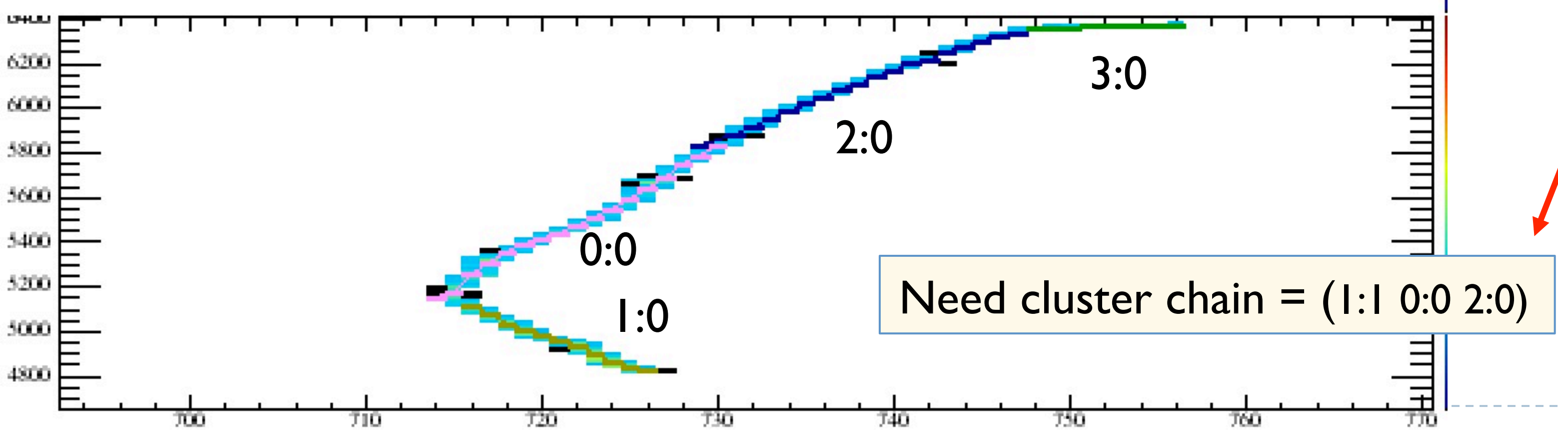


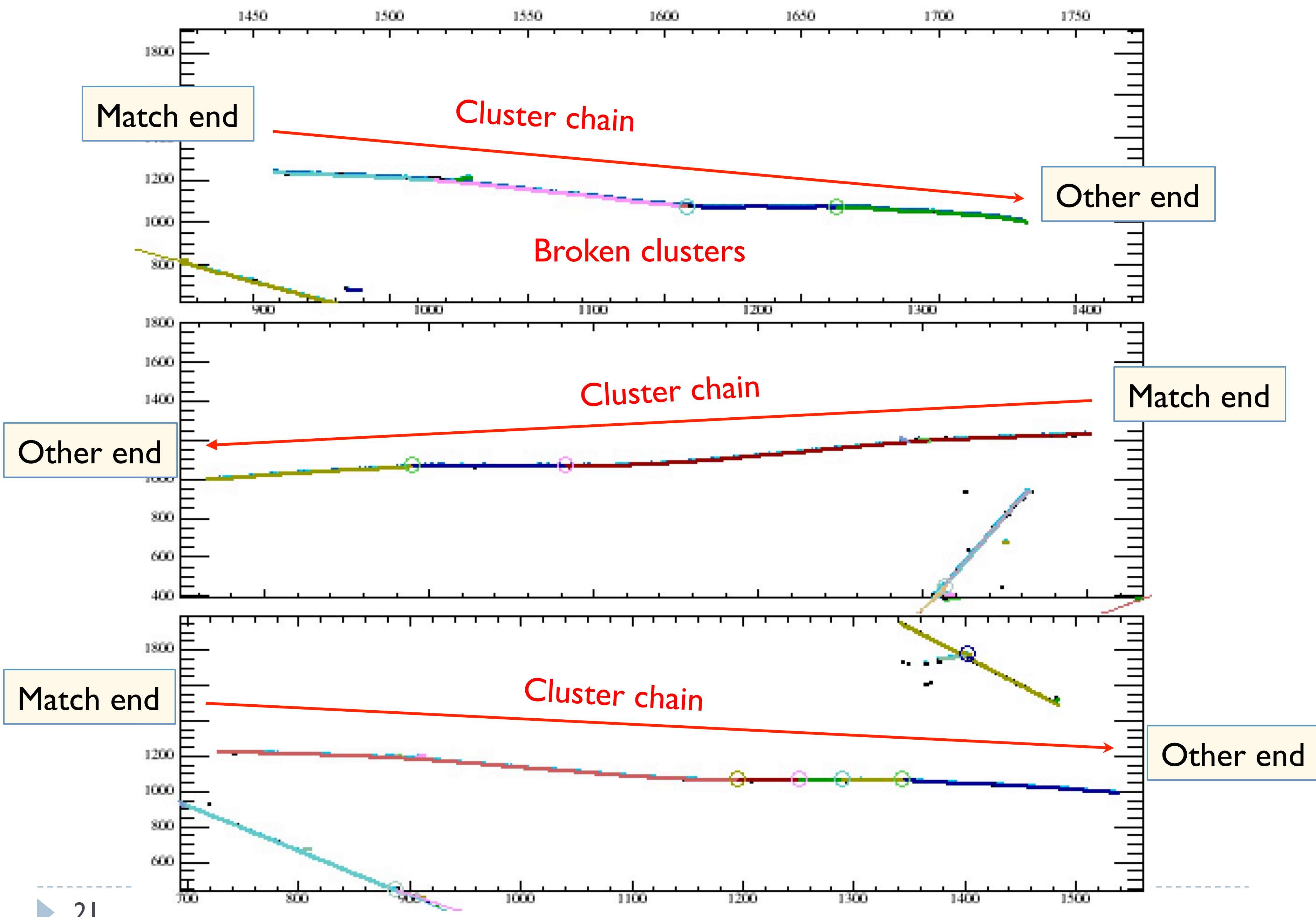
>>>>>>>>>> Cluster chains in Plane 0

ipl	ccl	Len	Chg	W0:T0	Ang0	Dir0	Vx0	CN0	W1:T1	Ang1	Dir1	Vx1	CN1	InTk	clChain:Order
0	0	44	29876	726:4832	-1.4	1	-1	1.5	747:6343	1.41	-1	-1	1.1	-1	1:1 0:0 2:0
0	1	8	900	748:6359	0.46	1	-1	1.4	756:6376	0.44	-1	-1	1.2	-1	3:0

>>>>>>>>>> Clusters in Plane 0

ipl	icl	Len	Chg	W0:T0	Ang0	Dir0	Vx0	CN0	W1:T1	Ang1	Dir1	Vx1	CN1	InTk	Brk0	Brk1
0	0	16	12444	714:5156	1.49	1	-1	0.0	730:5832	1.49	-1	-1	1.1	-1	1	2
0	1	10	10846	716:5116	-1.44	-1	-1	1.2	726:4832	-1.39	1	-1	1.5	-1	0	-1
0	2	18	6586	729:5835	1.45	1	-1	0.0	747:6343	1.41	-1	-1	1.1	-1	0	-1
0	3	8	900	748:6359	0.46	1	-1	1.4	756:6376	0.44	-1	-1	1.2	-1	-1	-1





Match end
Cluster chain
Other end
Broken clusters
Cluster chain
Match end
Other end
Match end
Cluster chain
Other end

CCTrackMaker_module
Matching between planes

```
// characterize the match between clusters in 2 or 3 planes
struct MatchPars {
  std::array<short, 3> Cls;
  std::array<unsigned short, 3> End;
  std::array<float, 3> Chg;
  short Vtx;
  float dWir;   // wire difference at the matching end
  float dAng;   // angle difference at the matching end
  float dX;     // X difference
  float Err;    // Wire,Angle,Time match error
  short oVtx;
  float odWir;  // wire difference at the other end
  float odAng;  // angle difference at the other end
  float odX;  // time difference at the other end
  float oErr;   // dAngle dX match error
};
// vector of many match combinations
std::vector<MatchPars> matcomb;
```

→ Potential 3D track

- SortMatches
  - Sort by increasing (Err + oErr)
  - Make tracks starting with the best cluster match combination, ignoring the ones that have already-used clusters
    - This fails if the correct match is ε > an incorrect match → not resilient

CCTrackMaker_module

SortMatches - New

- Method: Find the set of cluster matching combinations that has the lowest total matching error for ALL clusters in the event with the fewest number of tracks (matches)
- Find the total length of all clusters (not used in a track) in all planes in the matcomb vector (matcombTotLen)
  - VtxMatch: total length of all clusters associated with a vertex
  - PlnMatch: total length of all clusters in the TPC
- Double loop over match combinations, starting with the best
  - After the first loop, find the total length of all clusters used (totLen)
  - Stop looping if fracLen = totLen / matcombTotLen > 99.9%
  - Calculate a total error = Σ(match errors) * Σ (matches) / fracLen
  - Make tracks using match combinations with the best total error

CCTrackMaker_module

Code Development Output

```
******* PrintClusters *********  Num_Clusters_in
vtx  Index    X       Y       Z  Pln0 Pln1 Pln2
  0      0  128.1   26.4  822.4    2    2    1
>>>>>>>>>> Cluster chains in Plane 0
ipl    ccl  Len     Chg    W0:T0      Ang0 Dir0   Vx0   CN0    W1:T1      Ang1 Dir1   Vx1   CN1   InTk  clChain:End
  0      0  600   69598  655:4706  -0.31    -1    -1   1.2 1255:3982  -0.31    1    -1   1.1      5    0:0
  0      1   15    2233  739:4603  -0.47    -1    -1   1.8  754:4572  -0.57    1    -1   2.0     17    1:0
  0      2   31    6171  762:4557  -0.46    -1    -1   1.6  794:4500  -0.61    1    -1   1.4     12    2:0 4:0
  0      3    2    1088  768:4542  -0.86    -1    -1   0.0  770:4530  -0.86    1    -1   0.0     -1    3:0
  0      4    4    1621  806:4493  -0.51    -1    -1   1.4  810:4480  -0.51    1    -1   1.5     14    5:0
  0      5    2     620  819:4511   0.98     1    -1   0.0  821:4522   0.98   -1    -1   0.0     -1    6:0
  0      6  470   74909  830:3297   1.05     1    -1   1.3 1300:6463   1.04   -1    -1   1.5     10    7:0
  0      7    7    1984  836:4386  -0.40    -1    -1   1.2  843:4375  -0.40    1    -1   2.0     -1    8:0
  0      8    4     810  839:4490  -0.73    -1    -1   1.4  843:4475  -0.73    1    -1   1.9     -1    9:0
  0      9   15    2371  840:4389  -0.73    -1    -1   2.4  856:4335  -0.89    1    -1   1.2     18    10:0 11:0
  0     10    2     309  853:3480  -0.40    -1    -1   0.0  855:3476  -0.40    1    -1   0.0     -1    12:0
  0     11    3     621  857:4465   0.52     1    -1   0.0  860:4472   0.52   -1    -1   0.0     -1    13:0
  0     12   65    4311  922:8104   0.49     1    -1   1.2  987:8234   0.49   -1    -1   1.3      9    14:0
  0     13    4     640  933:4213  -0.39    -1    -1   1.3  937:4207  -0.44    1    -1   2.7     -1    15:0
  0     14    4     632  961:4166   0.06     0    -1   3.5  965:4168   0.01    0    -1   3.0     -1    16:0
  0     15    3     516  966:8190  -0.52    -1    -1   0.0  969:8184  -0.52    1    -1   0.0     -1    17:0
  0     16    2     482  987:4340   0.59     1    -1   0.0  989:4345   0.59   -1    -1   0.0     -1    18:0
  0     17    5    1186 1029:4255  -1.06    -1    -1   2.9 1034:4221  -1.14    1    -1   3.9     -1    19:0
  0     18    4     866 1069:4214   0.20     1    -1   4.4 1073:4218   0.32   -1    -1   5.1     -1    20:0
  0     19   13    2646 1131:4125  -0.65    -1    -1   0.0 1146:4091  -0.40    1    -1   2.5     -1    21:0 22:0
```

Match end

Other end

Cluster chain indices

SortMatches

ii	im	Vx	Err	dW	dA	dX	oVx	oErr	odW	odA	odX	Asym	icl	jcl	kcl
0	16	-1	0.08	0.8	-0.01	0.07	-1	0.01	-0.2	-0.00	-0.00	0.000	0:24:0	1:28:1	2:35:0
1	31	-1	0.43	-3.8	-0.06	-0.07	-1	0.08	1.2	-0.01	-0.10	0.000	0:53:1	1:35:0	2:55:0
2	28	-1	0.51	0.8	-0.09	0.16	-1	0.28	0.8	0.05	-0.06	0.000	0:40:0	1:47:0	2:50:0
3	23	-1	0.04	-0.2	0.01	-0.08	-1	0.74	4.8	0.12	0.22	0.000	0:53:0	1:35:1	2:55:1
4	17	-1	0.01	-0.2	-0.00	-0.00	-1	0.08	0.8	-0.01	0.07	0.000	0:24:1	1:28:0	2:35:1
5	0	-1	0.01	-0.2	-0.00	0.59	-1	0.05	0.8	-0.01	-0.67	0.001	0:0:0	1:0:1	2:5:0
6	15	-1	0.11	1.8	-0.00	-0.00	-1	0.07	-1.2	-0.00	-0.06	0.000	0:23:1	1:37:0	2:37:0
7	14	-1	0.07	-1.2	-0.00	-0.06	-1	0.11	1.8	-0.00	-0.00	0.000	0:23:0	1:37:1	2:37:1
8	13	-1	0.04	-0.2	-0.03	0.93	-1	0.08	0.8	-0.04	-3.31	0.000	0:22:1	1:6:0	2:14:1
9	12	-1	0.05	-0.2	-0.01	-0.02	-1	0.05	0.8	-0.00	-0.00	0.000	0:21:1	1:9:0	2:19:0
10	11	-1	0.05	0.8	-0.00	-0.00	-1	0.05	-0.2	-0.01	-0.02	0.000	0:21:0	1:9:1	2:19:1
11	10	-1	0.11	-0.2	0.12	2.77	-1	0.13	0.8	0.12	-3.90	0.000	0:12:1	1:34:0	2:13:1
12	1	-1	0.05	0.8	-0.01	-0.67	-1	0.01	-0.2	-0.00	0.59	0.001	0:0:1	1:0:0	2:5:1
13	7	-1	0.27	1.8	0.08	38.14	-1	0.08	0.8	0.05	-5.02	0.001	0:6:1	1:1:0	2:12:0
14	6	-1	0.07	0.8	0.05	-5.02	-1	0.14	1.8	0.08	38.14	0.001	0:6:0	1:1:1	2:12:1
15	22	-1	0.27	-0.2	0.04	-0.58	-1	1.11	11.8	-0.13	-0.55	0.007	0:42:0	1:26:1	2:28:1
16	4	-1	1.04	0.8	0.25	0.27	-1	1.47	-2.2	0.36	-2.44	0.006	0:2:0	1:16:1	2:4:1
17	18	-1	2.01	0.8	0.27	-0.59	-1	0.14	-1.2	-0.02	0.03	0.014	0:31:0	1:39:1	2:38:0
18	19	-1	0.14	-1.2	-0.02	0.03	-1	2.01	0.8	0.27	-0.59	0.014	0:31:1	1:39:0	2:38:1
19	20	-1	0.29	0.8	-0.04	-0.61	-1	2.43	-3.2	0.35	-1.19	0.038	0:33:1	1:42:0	2:40:0
20	24	-1	2.48	1.8	0.44	-0.04	-1	0.55	-2.2	0.09	-0.05	0.001	0:4:1	1:11:0	2:1:0
21	30	-1	0.14	0.8	0.01	0.07	-1	4.73	-36.2	-0.38	26.98	0.058	0:36:1	1:46:1	2:49:1
22	9	-1	3.00	-1.0	-1.00	-0.03	-1	3.00	0.0	0.00	0.00	0.000	0:12:0	1:34:1	2:-1:0
23	29	-1	3.04	-1.0	-1.00	0.40	-1	3.27	0.0	0.00	-0.14	0.737	0:44:1	1:-1:0	2:26:1
24	2	-1	3.00	-1.0	-1.00	-0.04	-1	3.01	0.0	0.00	-0.01	0.021	0:1:0	1:19:1	2:-1:0
25	25	-1	3.01	-1.0	-1.00	-0.10	-1	3.00	0.0	0.00	-0.00	0.016	0:-1:0	1:11:1	2:1:1
26	3	-1	3.40	-2.2	1.30	-2.23	-1	2.65	-5.2	1.00	0.10	0.039	0:1:1	1:19:0	2:6:0
27	5	-1	2.48	-0.2	-0.31	-0.24	-1	4.72	4.8	0.60	2.46	0.013	0:2:1	1:16:0	2:3:0
28	8	-1	4.08	3.8	-0.59	0.30	-1	4.28	2.8	-0.62	-0.09	0.004	0:9:0	1:7:1	2:2:1
29	26	-1	8.65	-0.2	0.90	-0.17	-1	1.02	9.8	0.04	2.32	0.174	0:41:0	1:26:1	2:26:1
30	27	-1	4.71	2.8	0.84	-0.04	-1	4.98	39.8	0.79	-1.81	0.000	0:40:0	1:47:0	2:49:1
31	21	-1	3.77	-0.2	0.46	0.12	-1	5.25	-55.2	0.22	28.09	0.013	0:36:1	1:47:0	2:50:0

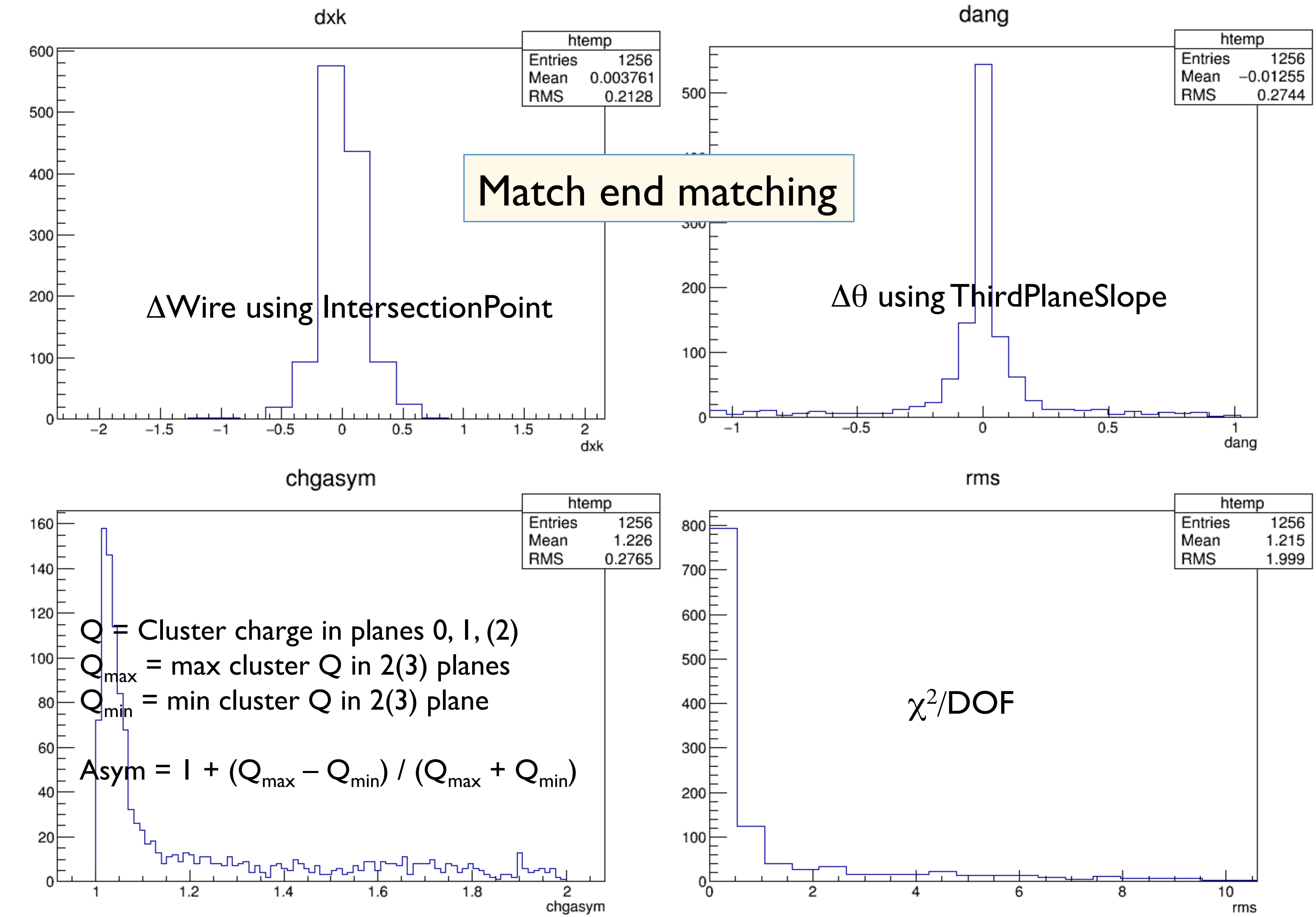
CCTrackMaker_module

- Produces

```
CCTrackMaker::CCTrackMaker(fhicl::ParameterSet const& pset)
{
  this->reconfigure(pset);
  produces< std::vector<recob::PFParticle>                     >();
  produces< art::Assns<recob::PFParticle, recob::Track>        >();
  produces< art::Assns<recob::PFParticle, recob::Cluster>      >();
  produces< art::Assns<recob::PFParticle, recob::Seed>         >();
  produces< art::Assns<recob::PFParticle, recob::Vertex>       >();
  produces< std::vector<recob::Vertex>                         >();
  produces< std::vector<recob::Track>                          >();
  produces< art::Assns<recob::Track,      recob::Hit>          >();
  produces<std::vector<recob::Seed>                            >();
}
```

- PFParticle convention
  - Neutrino PDGCode = 14
  - Neutrino Primary particles PDGCode = 2212
  - Neutrino Secondary particles PDGCode = 211
  - Cosmic rays PDGCode = 13
    - No attempt to associate delta-rays with muons

Match end matching



ΔWire using IntersectionPoint

$\Delta\theta$ using ThirdPlaneSlope

Q = Cluster charge in planes 0, 1, (2)

Q_{max} = max cluster Q in 2(3) planes

Q_{min} = min cluster Q in 2(3) plane

$\text{Asym} = 1 + (Q_{max} - Q_{min}) / (Q_{max} + Q_{min})$

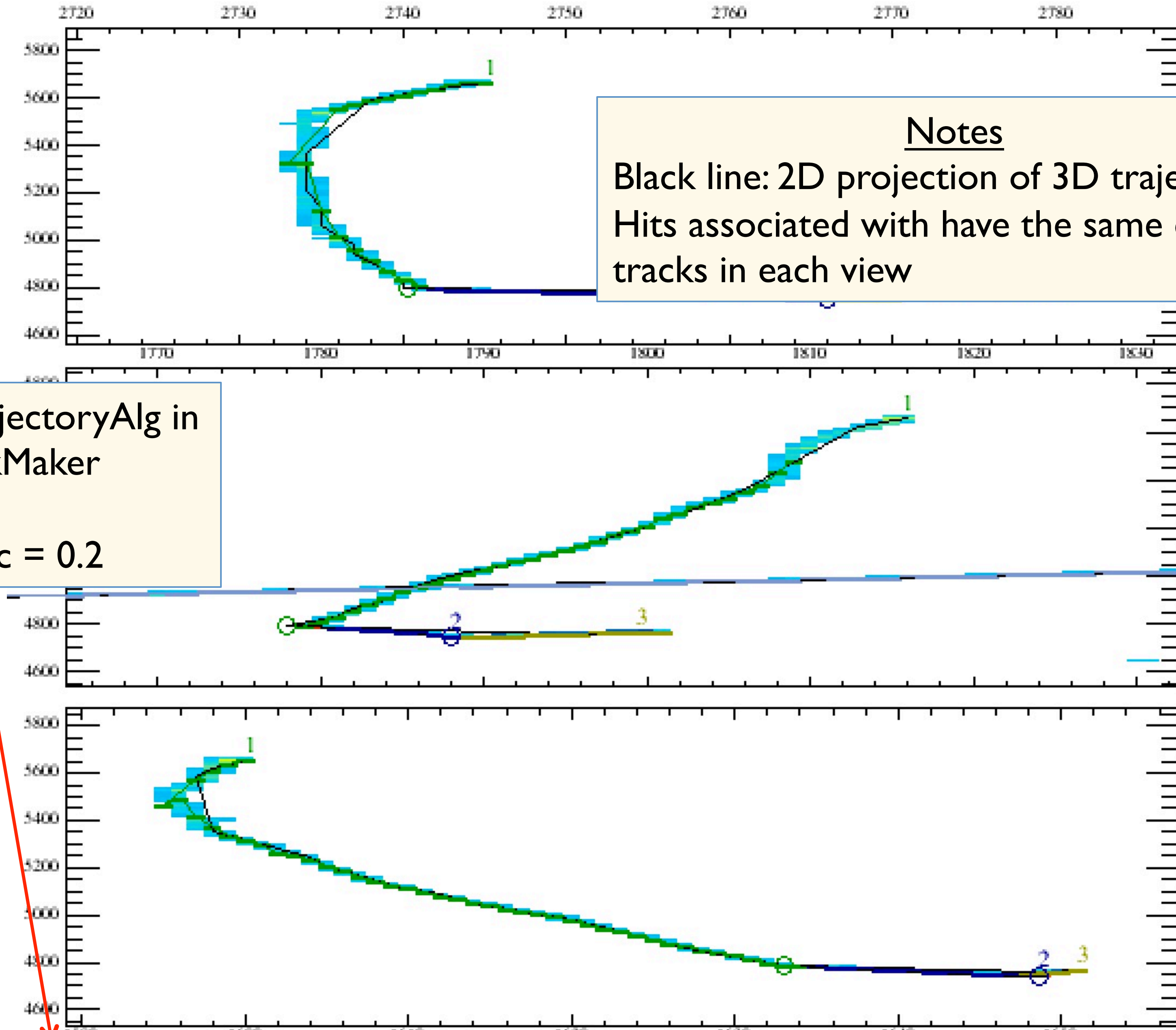
χ^2/DOF

TrackTrajectoryAlg - *Update*

- Reminder:
  - Finds a 3D trajectory using an ordered collection of hits in two or three planes
  - Method: Fit sub-collections of hits at similar X positions using TrackLineFitAlg to create trajectory points
  - Intended to be a simpler, faster alternative to Kalman fit module
- Problems with the current version
  - Failed too often
  - Flawed implementation for using the hit charge in lieu of X to create trajectory points
- New
  - Abandon charge method
  - Use a simpler scheme for making sub-collections
  - ShortTrackTrajectory finds trajectory endpoints for short tracks or if a failure occurs in the main algorithm
  - Breaking change to interface

```
void TrackTrajectoryAlg::TrackTrajectory(std::array<std::vector<geo::WireID>,3> trkWID,
                                         std::array<std::vector<double>,3> trkX,
                                         std::array<std::vector<double>,3> trkXErr,
                                         std::vector<TVector3>& TrajPos, std::vector<TVector3>& TrajDir)
{

  // Make a track trajectory (position, direction) and return it in the TrajPos
  // and TrajDir vectors. The track hits are received as 3 vectors (one vector per wire plane)
  // of Wire ID's, hit X and X position errors. The X position errors are used to 1) determine
  // the significance of the X difference between the beginning and end of the track trajectory
  // 2) determine the number of trajectory points and 3) weight the track line fit in TrackLineFitAlg.
  // This code assumes that hits at each end (e.g. trkXW[Plane][0]) of the vectors define the end
  // points of the trajectory. The ordering of planes in the array is irrelevant since the
  // plane number is extracted from the WireID. This algorithm will return with a failed condition
  // (TrajPos, TrajDir size = 0) if there are fewer than 2 planes with hits at each end that are less
  // than 5 * trkXErr apart. Valid and invalid conditions are shown schematically below where a . represents
  // hits that are separated by X values > 5 * trkXErr
  //
  //      minX            maxX             maxX            minX          minX            maxX
  // Pln0 ....................    Pln0 ....................     Pln0 ...................
  // Pln1 ...............         Pln1 ....................     Pln1
  // Pln2 ....................    Pln2 ................         Pln2 ...................
  // VALID                        VALID                         VALID - no hits in one plane is OK
  //
  //      minX            maxX
  // Pln0 .................
  // Pln1 ...............
  // Pln2 ....................
  // NOT VALID - Only one plane has a hit at MaxX
```



Notes

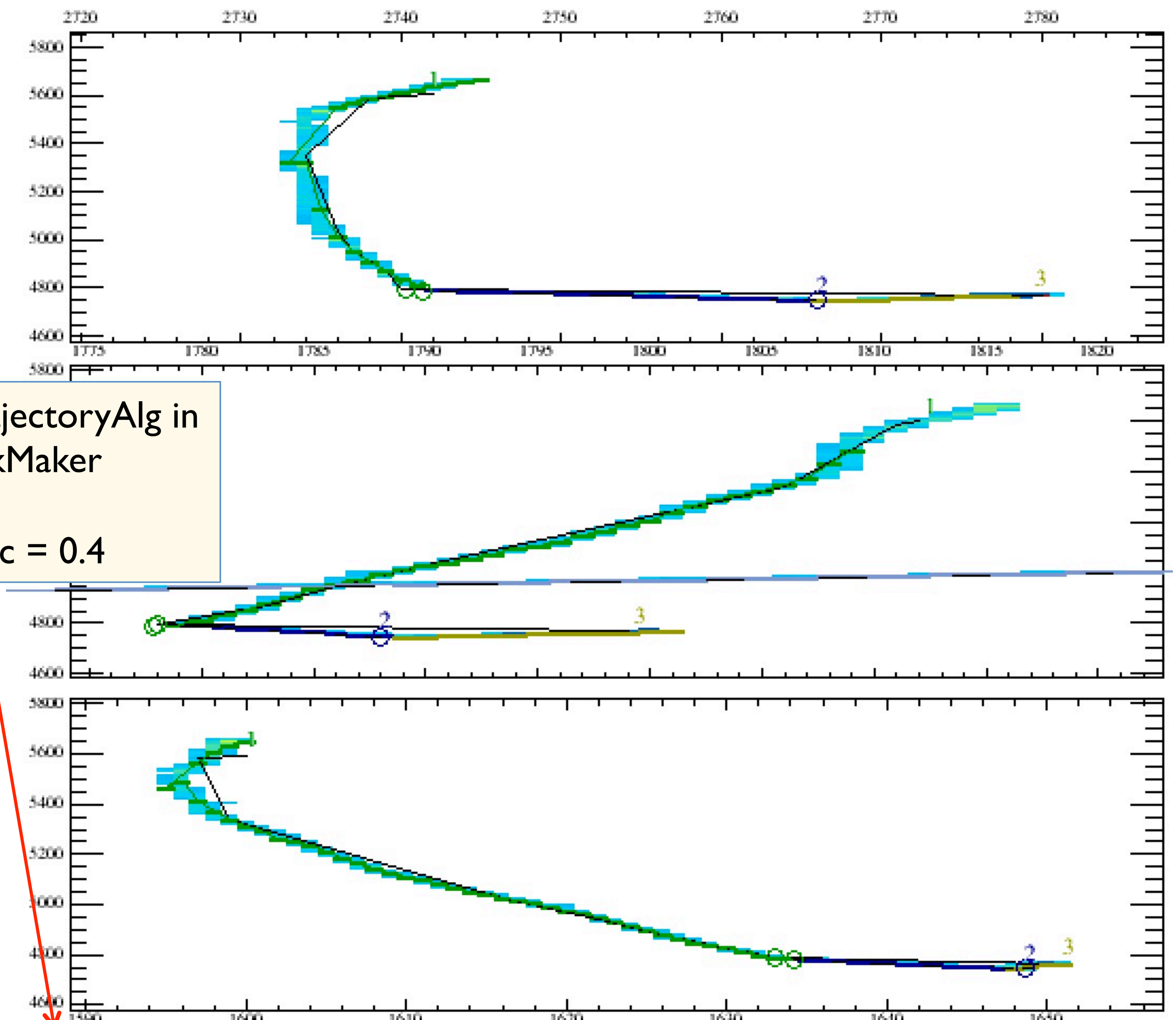
Black line: 2D projection of 3D trajectory

Hits associated with have the same color tracks in each view

TrackTrajectoryAlg in CCTrackMaker

HitErrFac = 0.2

```
>>>>>>>>>>> Tracks
trk  ID  Proc nht nTrj  sX     sY     sZ     eX     eY     eZ   sVx eVx ChgOrder dirZ Mom PDG     ClsIndices
  0  -1    2   94  23 127.5   26.0  822.4  197.1   38.3  823.8   0  -1   0       0.014   0 2212   12 30 55 116 136 179 177
  1  -2    2   45   6 127.5   26.0  822.4  124.0   25.0  830.1   0   1   0      -0.863   0 2212   11 70 126
  2  -3    2   30   3 127.5   26.0  822.4  125.6   26.6  834.2   0  -1   0       0.899   0 2212   26 69 125
```



TrackTrajectoryAlg in CCTrackMaker

HitErrFac = 0.4

```
>>>>>>>>>>> Tracks
```

trk	ID	Proc	nht	nTrj	sX	sY	sZ	eX	eY	eZ	sVx	eVx	ChgOrder	dirZ	Mom	PDG	ClsIndices
0	-1	2	94	7	127.5	26.0	822.4	192.4	37.7	822.8	0	-1	0	0.000	0	2212	12 30 55 116 136 179 177
1	-2	2	45	3	127.5	26.0	822.4	124.0	25.0	830.1	0	1	0	-0.874	0	2212	11 70 126
2	-3	2	30	2	127.5	26.0	822.4	125.6	26.6	834.2	0	-1	0	0.899	0	2212	26 69 125

VertexFitAlg - *New*

- Inputs
  - Calling routine reconstructs 3D tracks from hits in 2(3) planes
    - Passes a subset of the track hit collection to VertexFixAlg: WireID, X, X error

```
void VertexFitAlg::VertexFit(std::vector<std::vector<geo::WireID>>& hitWID,
                             std::vector<std::vector<double>>& hitX,
                             std::vector<std::vector<double>>& hitXErr,
                             TVector3& VtxPos, TVector3& VtxPosErr,
                             std::vector<TVector3>& TrkDir, std::vector<TVector3>& TrkDirErr,
                             float& ChiDOF)
```

- Information put in a global struct so that TMinuit can see it →
- Outputs
  - Vertex position + errors
  - Fitted track direction vectors + errors
  - Chisq/DOF

VertexFitMinuitStruct.h

```
struct VertexFitMinuitStruct {

  unsigned short TPC;
  unsigned short Cstat;
  unsigned short NPlanes;
  double WirePitch;
  std::array<double, 3> XFactor;                  // The denominator factor in ConvertXToTicks
  std::array<double, 3> TickOff;                  // The tick offset in ConvertXToTicks
  std::array<double, 3> OrthY;
  std::array<double, 3> OrthZ;
  std::array<double, 3> FirstWire;                // the FirstWireProj in WireCoordinate
  TVector3 VtxPos;                // Vertex position (detector units)

  std::vector<std::vector<double>> HitX; // hit X
  //   track      X
  std::vector< std::vector<double>> HitXErr; // hit X errors
  std::vector<std::vector<unsigned short>> Plane;
  std::vector<std::vector<unsigned short>> Wire;
  //   track
  std::vector<TVector3> Dir;
  std::vector<TVector3> DirErr;
  double DoF;
  float ChiDoF;                                 // fit Chisq/DOF

};
```

```
void VertexFitAlg::fcnVtxPos(Int_t &, Double_t *, Double_t &fval, double *par, Int_t flag)
{
  // Minuit function for fitting the vertex position and vertex track directions

  fval = 0;
  double vWire, DirX, DirY, DirZ, DirU, dX, dU, arg;
  unsigned short ipl, lastpl, indx;

  for(unsigned short itk = 0; itk < fVtxFitMinStr.HitX.size(); ++itk) {
    lastpl = 4;
    // index of the track Y direction vector. Z direction is the next one
    indx = 3 + 2 * itk;
    for(unsigned short iht = 0; iht < fVtxFitMinStr.HitX[itk].size(); ++iht) {
      ipl = fVtxFitMinStr.Plane[itk][iht];
      if(ipl != lastpl) {
        // get the vertex position in this plane
        // vertex wire number in the Detector coordinate system (equivalent to WireCoordinate)
        //vtx wir = vtx Y  * OrthY                + vtx Z  * OrthZ                    - wire offset
        vWire = par[1] * fVtxFitMinStr.OrthY[ipl] + par[2] * fVtxFitMinStr.OrthZ[ipl] - fVtxFitMinStr.FirstWire[ipl];
//          if(flag == 1) mf::LogVerbatim("VF")<<"fcn vtx "<<par[0]<<" "<<par[1]<<" "<<par[2]<<" vWire "<<vWire<<" OrthY
  "<<fVtxFitMinStr.OrthY[ipl]<<" OrthZ "<<fVtxFitMinStr.OrthZ[ipl];
        lastpl = ipl;
      } // ipl != lastpl
      DirY = par[indx];
      DirZ = par[indx + 1];
      // rotate the track direction DirY, DirZ into the wire coordinate of this plane. The OrthVectors in ChannelMapStandardAlg
      // are divided by the wire pitch so we need to correct for that here
      DirU = fVtxFitMinStr.WirePitch * (DirY * fVtxFitMinStr.OrthY[ipl] + DirZ * fVtxFitMinStr.OrthZ[ipl]);
      // distance (cm) between the wire and the vertex in the wire coordinate system (U)
      dU = fVtxFitMinStr.WirePitch * (fVtxFitMinStr.Wire[itk][iht] - vWire);
      if(std::abs(DirU) < 1E-3 || std::abs(dU) < 1E-3) {
        // vertex is on the wire
        dX = par[0] - fVtxFitMinStr.HitX[itk][iht];
      } else {
        // project from vertex to the wire. We need to find dX/dU so first find DirX
        DirX = 1 - DirY * DirY - DirZ * DirZ;
        // DirX should be > 0 but the bounds on DirY and DirZ are +/- 1 so it is possible for a non-physical result.
        if(DirX < 0) DirX = 0;
        DirX = sqrt(DirX);
        // Get the DirX sign from the relative X position of the hit and the vertex
        if(fVtxFitMinStr.HitX[itk][iht] < par[0]) DirX = -DirX;
        dX = par[0] + (dU * DirX / DirU) - fVtxFitMinStr.HitX[itk][iht];
      }
      arg = dX / fVtxFitMinStr.HitXErr[itk][iht];
//        if(flag == 1) mf::LogVerbatim("VF")<<"fcn itk "<<itk<<" iht "<<iht<<" ipl "<<ipl<<" DirX "<<DirX<<" DirY "<<DirY<<" DirZ "<<DirZ
//        <<" DirU "<<DirU<<" W "<<fVtxFitMinStr.Wire[itk][iht]<<" X "<<fVtxFitMinStr.HitX[itk][iht]<<" dU "<<dU<<" dX "<<dX<<" arg "<<arg;
      fval += arg * arg;
    } // iht
```

Summary

- Testing done on a mac using v04_15_00 mavericks distribution
- Further improvements
  - Request feedback from users
  - Matching shower-like clusters in CCTrackMaker