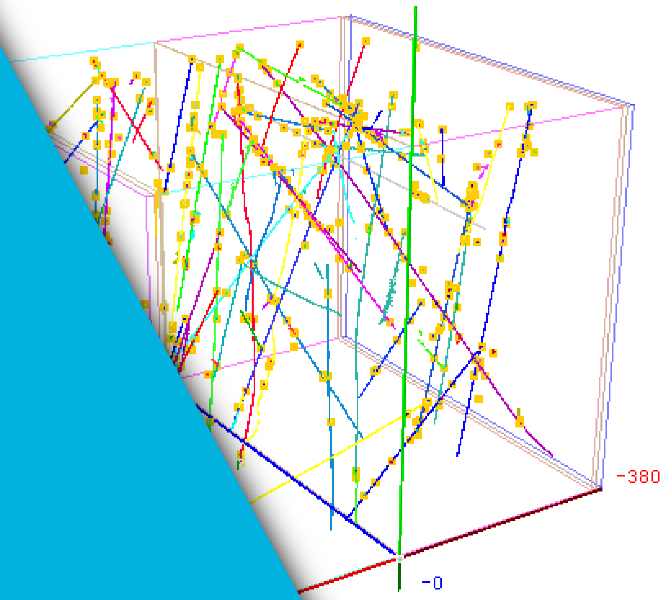
# Debugging reconstruction

(Exercise)

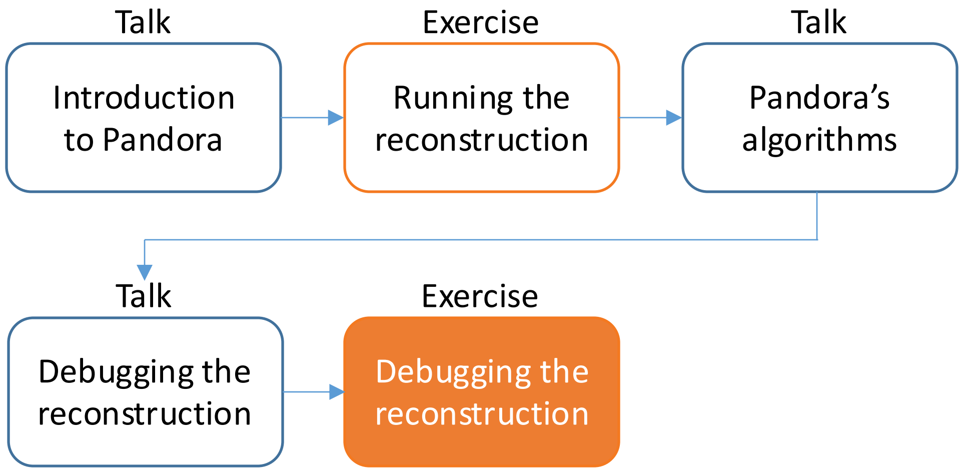
Andy Chappell and Isobel Mawby for the Pandora team

29/10/2023

9th UK LArTPC Software and Analysis Workshop



## Reconstruction session


Talk
Introduction to Pandora
Exercise
Running the reconstruction
Talk
Pandora's algorithms
Talk
Debugging the reconstruction
Exercise
Debugging the reconstruction

Credit: These slides build on previous LArSoft workshop slides by Andrew Smith-Jones

Key references: Pandora ProtoDUNE paper
Pandora MicroBooNE paper

## Goals

- This session scheduled for 1 hour 20 minutes
- Main goal – Identify the source of reconstruction failures
  - Add visualizations at key points of the reconstruction chain to understand how the reconstruction proceeds
  - Investigate the behaviour of relevant algorithms to understand how they work
- Secondary goal – Fix events
  - Identify algorithms that might address the issues and add those algorithms into the chain
  - Attempt to tune key parameters in XML, as needed
  - Continue to use intermediate visualization to understand what your changes have done
- Please don't worry if you don't fix the issues
  - Even if you fix an error in one place, your efforts might be undone later in the reconstruction chain
    - The reconstruction algorithms interact with each other
  - You may be able to fix parts of an event, but not others
  - Reconstruction is hard

## Investigate the baseline reconstruction output

## An aside on enabling the Pandora event display

- In this session we'll be using a custom configuration, but the typical top-level Pandora configuration file will be called something like PandoraSettings_Master_SBND.xml
- If all you want to do is enable the Pandora event display in an otherwise standard configuration you can copy this file locally and enable Pandora Monitoring with the following modification:

```
<pandora>
    <!-- GLOBAL SETTINGS -->
    <IsMonitoringEnabled>true</IsMonitoringEnabled>
    ...
```

## A new event

- We'll be using custom neutrino events for this task, rather than the 1 muon, 1 proton events you've been looking at so far (don't forget to replace the wildcard):

```
/mnt/gridpp/poolhomes/PPEGroup/LAR24/reconstruction/sbnd_nu_reco1_*.root
```

- Make sure to be in the directory you've been working in so far for the reconstruction tutorials and then copy the configuration files we'll be using for this session:

```
$ cd $MRB_TOP/reco/config
$ cp /mnt/gridpp/poolhomes/PPEGroup/LAR24/reconstruction/PandoraSettings_Master_Simple.xml .
$ cp /mnt/gridpp/poolhomes/PPEGroup/LAR24/reconstruction/PandoraSettings_Neutrino_Simple.xml .
```

- These files contain a greatly simplified reconstruction workflow, where many Pandora algorithms have been removed, to introduce reconstruction failures to these events

If you closed your terminal since the last session, don't forget to set everything up again! You will also need to export your FHICL_FILE_PATH again!

## Ensuring your custom files are found

- At the moment, Pandora won’t know about our custom configuration files, so we need to add our config directory to the FW_SEARCH_PATH so Pandora knows where to look for it (you might already have this in a setup script) and do the same for the FHICL_FILE_PATH:

```
$ export FW_SEARCH_PATH=$MRB_TOP/reco/config:$FW_SEARCH_PATH
$ export FHICL_FILE_PATH=$MRB_TOP/reco/config:$FHICL_FILE_PATH
```

- Now we need to set up a new FHiCL file to run our custom configuration…

## Writing a FHiCL file to run custom reconstruction

- Let’s make a new FHiCL file that just runs the reconstruction using our custom XML configuration and produces an output file without the default timestamp tag

```
$ vim reco_driver.fcl
```

- Add the lines below to reco_driver.fcl, save and close:

```
#include "standard_reco2_sbnd.fcl"

# Use our custom settings file
physics.producers.pandora.ConfigFile: "PandoraSettings_Master_Simple.xml"
```

Start from the standard reco chain

Point to our new XML settings file
This is a very convenient way to customize FHiCL – inherit and minimally modify

- Run Pandora over the event and look at the reconstruction output

```
$ lar -c reco_driver.fcl -s /mnt/gridpp/poolhomes/PPEGroup/LAR24/reconstruction/sbnd_nu_reco1.root -n 1 --nskip 2
```

When copy-pasting commands there can be strange character conversions, particularly for hyphens, these often show up as an inability to find a config file. Try deleting the hyphens and just typing them

## A brief introduction to the Pandora event display

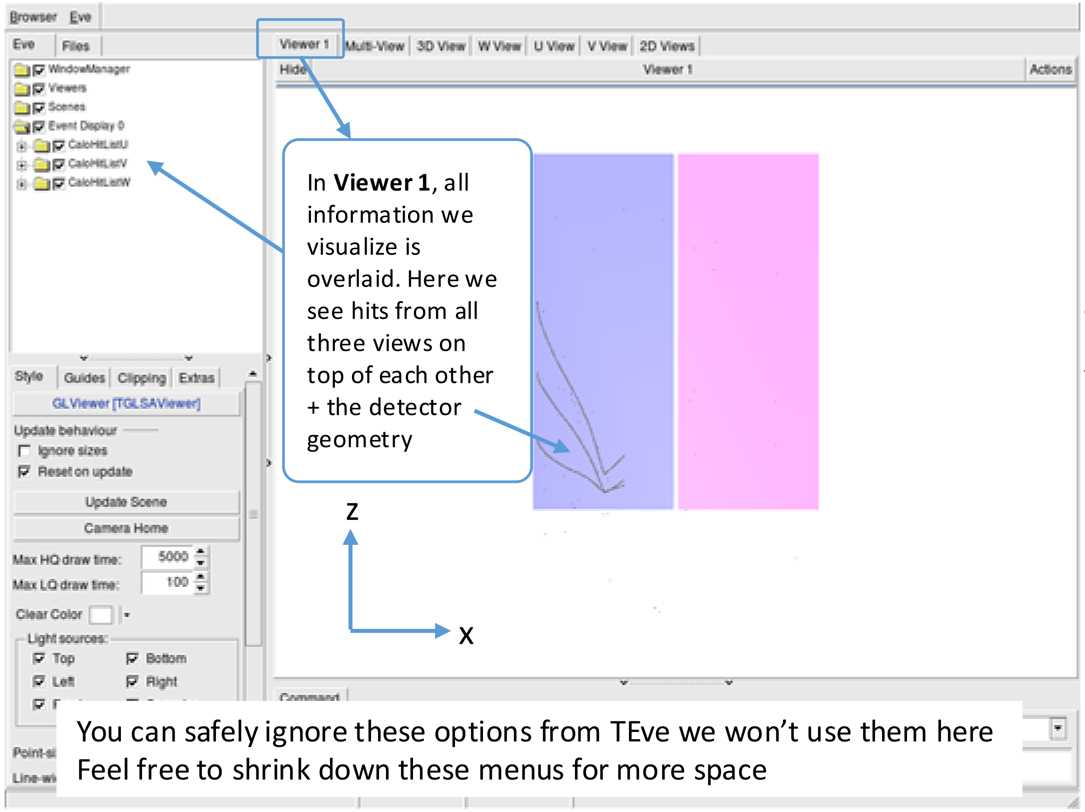
# Understanding the Pandora event display

Every time the visual monitoring algorithm runs, we get a new event display (enumerated from zero) →

Try checking and unchecking the boxes to turn on and off the hits from each of the views
☑ CaloHitListU
☑ CaloHitListV
☑ CaloHitListW

The 2D hit coordinates are stored in Pandora as 3D coordinates (X, Y, Z)

X = drift time coordinate
Y = 0
Z = wire number coordinate



☐ Ignore sizes
☑ Reset on update
Max HQ draw time: 5000
Max LQ draw time: 100

In **Viewer 1**, all information we visualize is overlaid. Here we see hits from all three views on top of each other + the detector geometry

You can safely ignore these options from TEve we won't use them here
Feel free to shrink down these menus for more space

Wheel up - zoom out
Wheel down - zoom in
Wheel press + drag - pan viewport

(Or whatever the mapping of these operations is for your laptop. Arrow keys can move the view around as well)

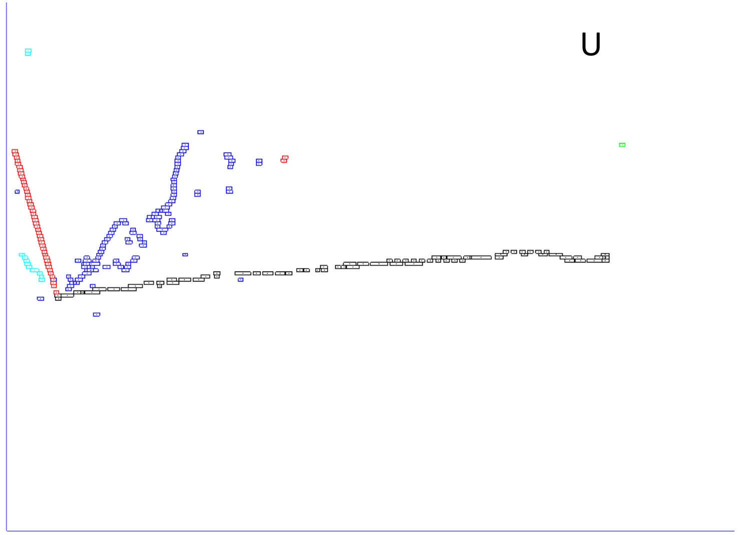
W - wireframe mode
R - return from wireframe mode

If you click in the terminal window and press Return ↲ Pandora will continue running

## An example event

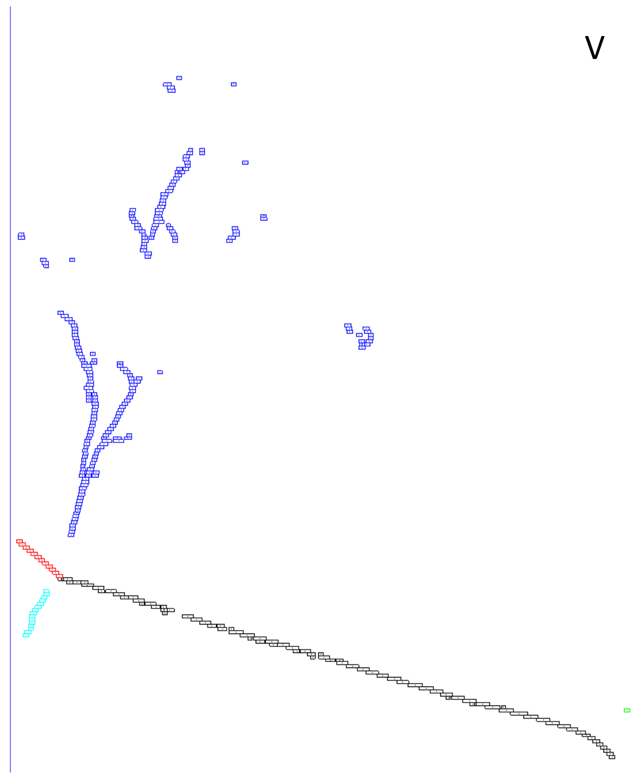
## The underlying event (1)

- We’ll highlight several events that are available to investigate in this session, but we’ll focus on one to outline the process
- Depicted to the right is the MC truth for this resonant pion production event in the U view (V and W on the next two slides)
- The primary particles in this event are
  - Muon (black)
  - Proton (red)
  - Leading photon (blue)
  - Sub-leading photon (cyan)
  - 5 very low energy photons (not all visible in this image)
    - Expect these to be very difficult to reconstruct under any circumstances



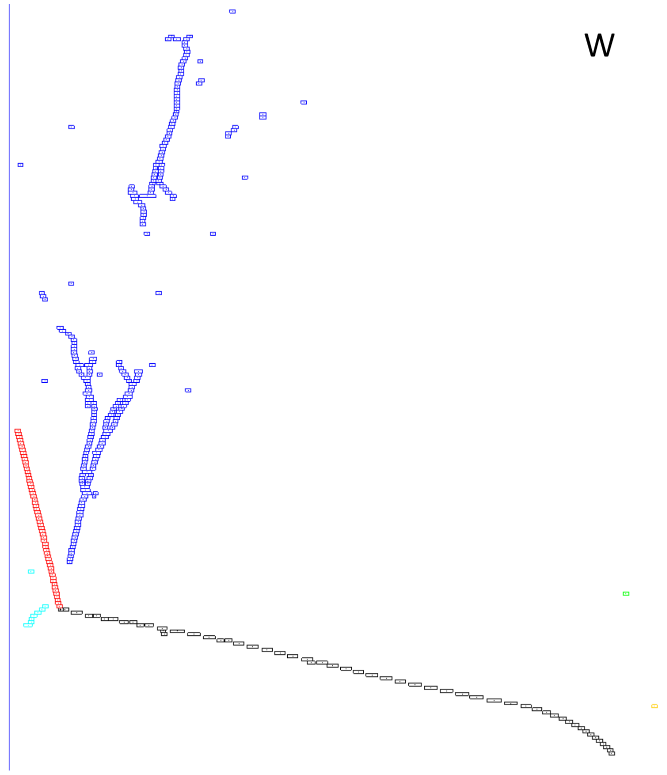
## The underlying event (2)

- We'll highlight several events that are available to investigate in this session, but we'll focus on one to outline the process
- Depicted to the right is the MC truth for this resonant pion production event in the V view (U and W on the preceding and following slides)
- The primary particles in this event are
  - Muon (black)
  - Proton (red)
  - Leading photon (blue)
  - Sub-leading photon (cyan)
  - 5 very low energy photons (not all visible in this image)
    - Expect these to be very difficult to reconstruct under any circumstances



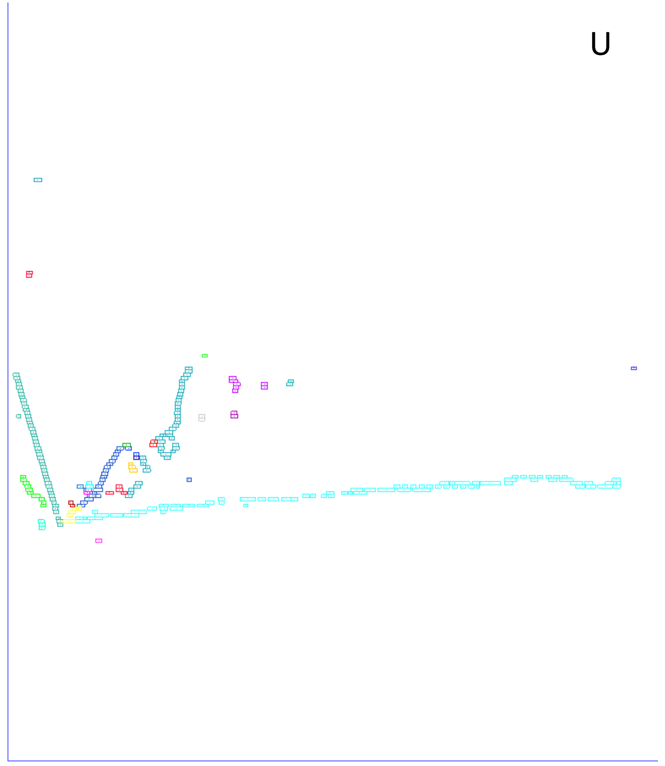
## The underlying event (3)

- We'll highlight several events that are available to investigate in this session, but we'll focus on one to outline the process
- Depicted to the right is the MC truth for this resonant pion production event in the W view (U and V on the previous two slides)
- The primary particles in this event are
  - Muon (black)
  - Proton (red)
  - Leading photon (blue)
  - Sub-leading photon (cyan)
  - 5 very low energy photons (not all visible in this image)
    - Expect these to be very difficult to reconstruct under any circumstances



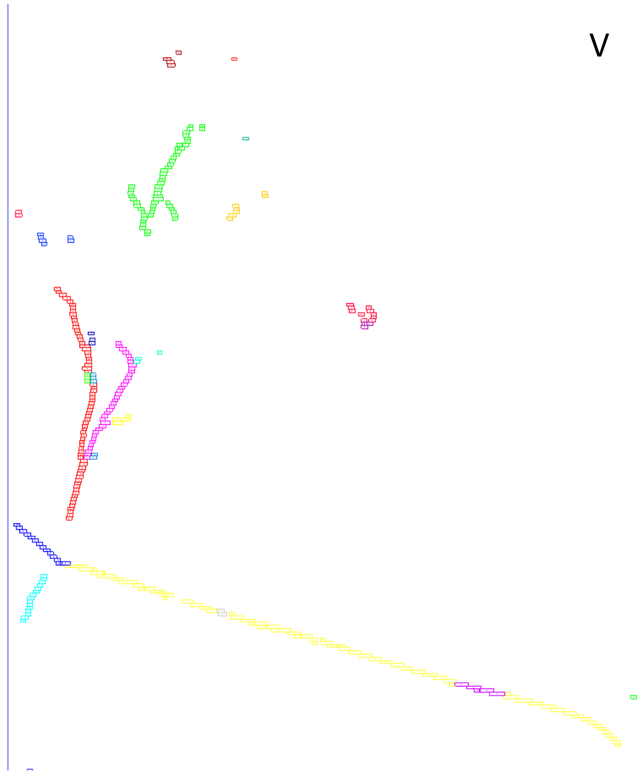
## Reconstructing the event (1)

- If we run our simplified reconstruction and consider the clusters in the U view, a few things are evident:
  - The proton and muon are well-reconstructed, with some minor hit confusion between the tracks at the vertex
  - The shower reconstruction is less successful
    - There is notable shower fragmentation
    - We'll ignore the remaining very low energy showers, albeit they have largely been reconstructed in 2D



## Reconstructing the event (2)

- In the V view, a similar picture is evident:
  - The proton is well reconstructed with some minor hit confusion at the vertex, where it steals a hit from the muon
  - The muon is mostly well reconstructed, but for a small fragments along the track
  - The shower reconstruction is less successful
    - The sub-leading shower is well reconstructed in this view
    - But notable shower fragmentation exists in the leading shower



## A few pointers (1)

- There are more than 100 algorithms available to Pandora's pattern recognition process, so to help narrow things down, we'll highlight a few potentially interesting algorithms that might help resolve some or all of these issues
- We won't go through the details now, but in the backup section of these slides are brief descriptions of what these algorithms do, with a focus on the 2D algorithms
- 2D clustering algorithms – these can be used after algorithms like LArTransverseExtension
  - LArCrossGapsAssociation, LArCrossGapsExtension, LArOvershootSplitting, LArBranchSplitting, LArKinkSplitting, LArTrackConsolidation, LArHitWidthClusterMerging
  - Note these algorithms run for each of U, V and W, and so anything you add must be replicated in each of the relevant sections
- 2D mop up algorithms – can be used after LArThreeDShowers
  - LArBoundedClusterMopUp, LArConeClusterMopUp, LArNearbyClusterMopUp

## A few pointers (2)

- There are more than 100 algorithms available to Pandora's pattern recognition process, so to help narrow things down, we'll highlight a few potentially interesting algorithms that might help resolve some or all of these issues
- We won't go through the details now, but in the backup section of these slides are brief descriptions of what these algorithms do, with a focus on the 3D algorithms
- 3D track algorithms – can be used after LArThreeDLongitudinalTracks
  - LArThreeDTrackFragments
- 3D recovery algorithms - can be used after LArThreeDShowers
  - LArVertexBasedPfoRecovery, LArParticleRecovery
- 3D mop up algorithms - can be used after LArThreeDHitCreation
  - LArSlidingConePfoMopUp, LArSlidingConeClusterMopUp, LArIsolatedClusterMopUp

## Understanding the reconstruction (1)

- We’re going to use Pandora’s event display to explore how the clustering proceeds in an effort to understand where we might want to intervene
- We’ll start by focusing on the 2D clustering algorithms: After the W view algorithms in the <!-- TwoDReconstruction --> block, add the following instance of the LArVisualMonitoring algorithm

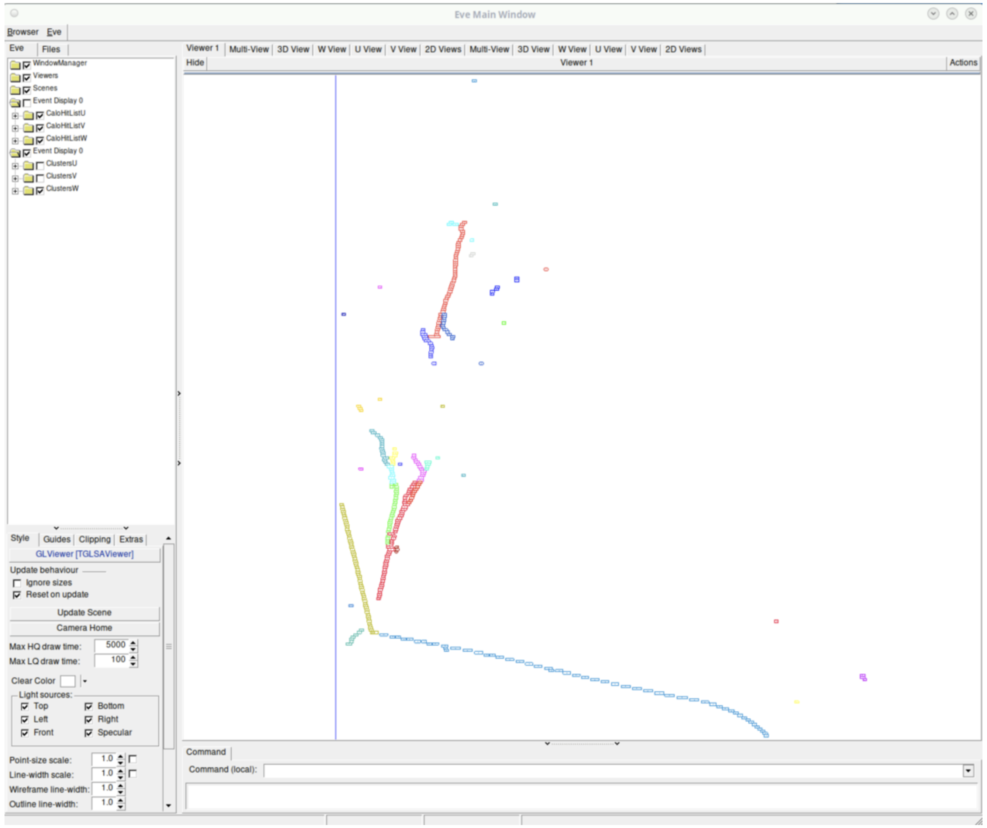
```
<!-- TwoDReconstruction -->
...
<algorithm type = "LArClusteringParent">
    <algorithm type = "LArTrackClusterCreation" description = "ClusterFormation"/>
    <InputCaloHitListName>CaloHitListW</InputCaloHitListName>
    ...
</algorithm>
...
<algorithm type = "LArTransverseExtension"/>

<algorithm type = "LArVisualMonitoring">
    <ClusterListNames>ClustersU ClustersV ClustersW</ClusterListNames>
    <ShowDetector>true</ShowDetector>
</algorithm>
```

- You may also want to set ShouldDisplayAlgorithmInfo to true at the beginning of both the master and neutrino XML, in order to keep track of where you are as you step through the event displays

## Understanding the reconstruction (2)

- Run Pandora and start to observe how the clustering proceeds
- Focus on the W view to begin with by deselecting the other views in the tree
- Note how small many of the clusters are
- Provisional clustering is track-centric and quite strict
  - Ambiguity or sharp deviations in trajectory limit the growth of clusters
  - This means the tracks in this event, and this view, are well reconstructed – there’s little doubt about how the tracks should grow
  - The showers are fragmented



## Understanding the reconstruction (3)

- Now add another instance of the LArVisualMonitoring algorithm, but this time just after the <!-- VertexAlgorithms --> block, shown below
- Here we'll look at the clusters in the W view, along with the selected Neutrino interaction vertex (which, being in 3D, only projects onto the W view)
  - The vertex is an important reference point for the reconstruction, so errors here can have knock on effects
  - In SBND, Pandora currently uses a BDT to select from a (potentially large) set of candidate vertices
  - In DUNE, Pandora employs a deep neural network to determine the vertex location – this is being tested for deployment at SBND

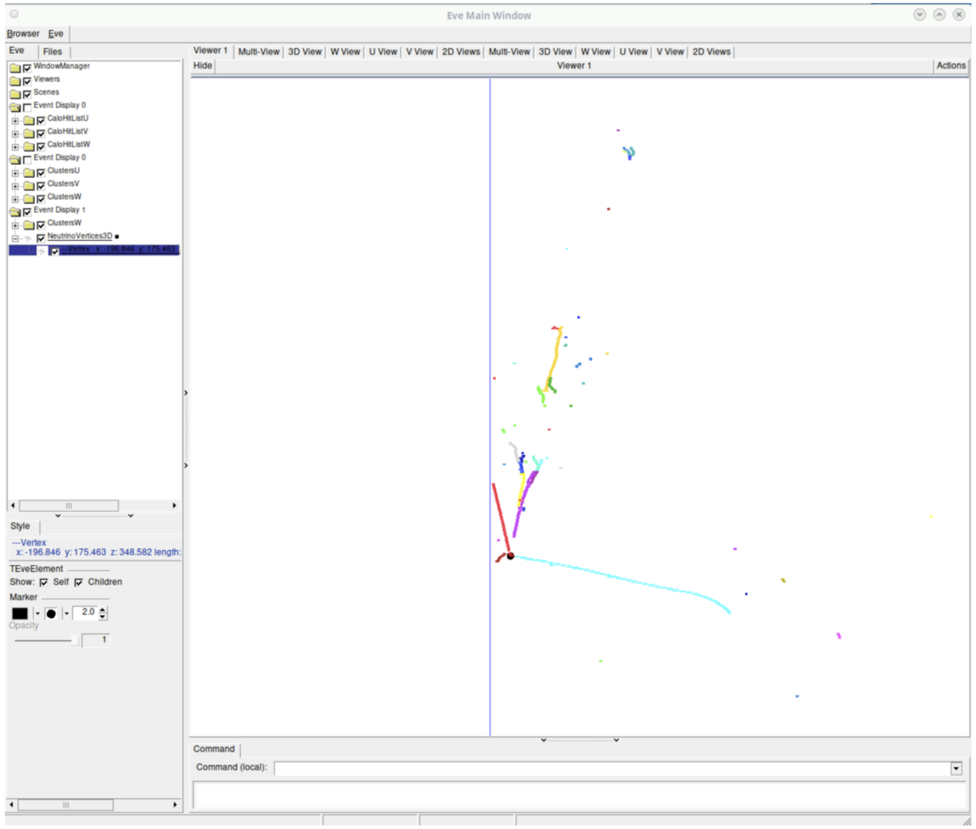
```
<!-- VertexAlgorithms -->
...
<algorithm type = "LArCutClusterCharacterisation">
    <InputClusterListNames>ClustersU ClustersV ClustersW</InputClusterListNames>
    <ZeroMode>true</ZeroMode>
</algorithm>

<algorithm type = "LArVisualMonitoring">
    <ClusterListNames>ClustersW</ClusterListNames>
    <VertexListNames>NeutrinoVertices3D</VertexListNames>
    <ShowDetector>true</ShowDetector>
</algorithm>
```

- Re-run Pandora to see this new event display (if you haven't removed it, you'll need to skip past the previous event display first)

## Understanding the reconstruction (4)

- You'll see the same set of clusters, but now the reconstructed neutrino interaction vertex as well
    - Pandora renders these vertices in yellow, which can be tricky to spot
    - If you expand the NeutrinoVertices3D node in the tree and select Vertex, you'll see options to change the colour and size of the marker, as we've done here
- Here the vertex appears well placed


Eve Main Window
Browser Eve
Eve Files
Viewer 1 Multi-View 3D View W View U View V View 2D Views Multi-View 3D View W View U View V View 2D Views
Hide Viewer 1 Actions
Command
Command (local):

WindowManager
Viewers
Scenes
Event Display 0
CaloHitListU
CaloHitListV
CaloHitListW
Event Display 0
ClustersU
ClustersV
ClustersW
Event Display 1
ClustersW
NeutrinoVertices3D
--Vertex x: -196.846 y: 175.463

Style
---Vertex
x: -196.846 y: 175.463 z: 348.582 length:
TEveElement
Show: [✓ Self] [✓ Children]
Marker
2.0
Opacity
1

## Understanding the reconstruction (5)

- Next we'll add another instance of the LArVisualMonitoring algorithm, now just after the <!-- ThreeDTrackAlgorithms --> block, shown below
- We'll focus on the W view again here, but all three views are available in the full visualization, because that's typically most useful

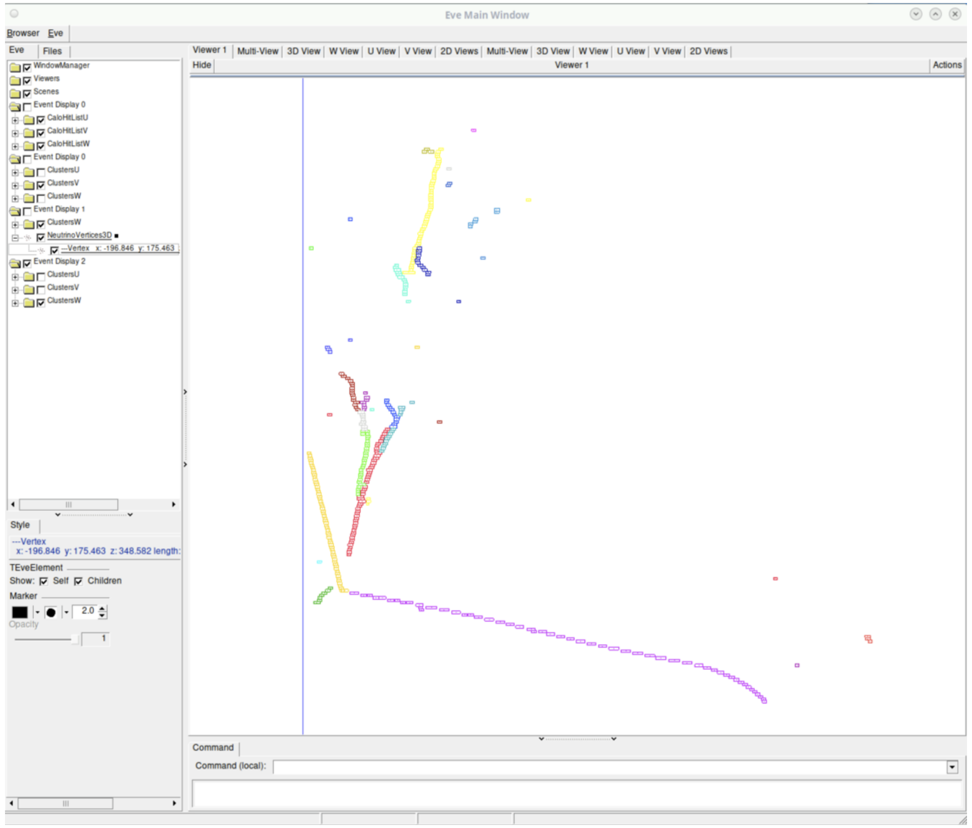
```
<!-- ThreeDTrackAlgorithms -->
...
<algorithm type = "LArThreeDLongitudinalTracks">
    <InputClusterListNameU>ClustersU</InputClusterListNameU>
    <InputClusterListNameV>ClustersV</InputClusterListNameV>
    <InputClusterListNameW>ClustersW</InputClusterListNameW>
    <OutputPfoListName>TrackParticles3D</OutputPfoListName>
    <TrackTools>
        <tool type = "LArClearLongitudinalTracks"/>
        <tool type = "LArMatchedEndPoints"/>
    </TrackTools>
</algorithm>

<algorithm type = "LArVisualMonitoring">
    <ClusterListNames>ClustersU ClustersV ClustersW</ClusterListNames>
    <ShowDetector>true</ShowDetector>
</algorithm>
```

- Re-run Pandora to see this new event display

## Understanding the reconstruction (6)

- Given the track reconstruction already looked quite complete before this point, we wouldn't expect to see much here



## Understanding the reconstruction (7)

- Next, visualize just after the <!– ThreeDShowerAlgorithms --> block, shown below
- We’ll focus on the W view again here, but all three views are available in the full visualization, because that’s typically most useful

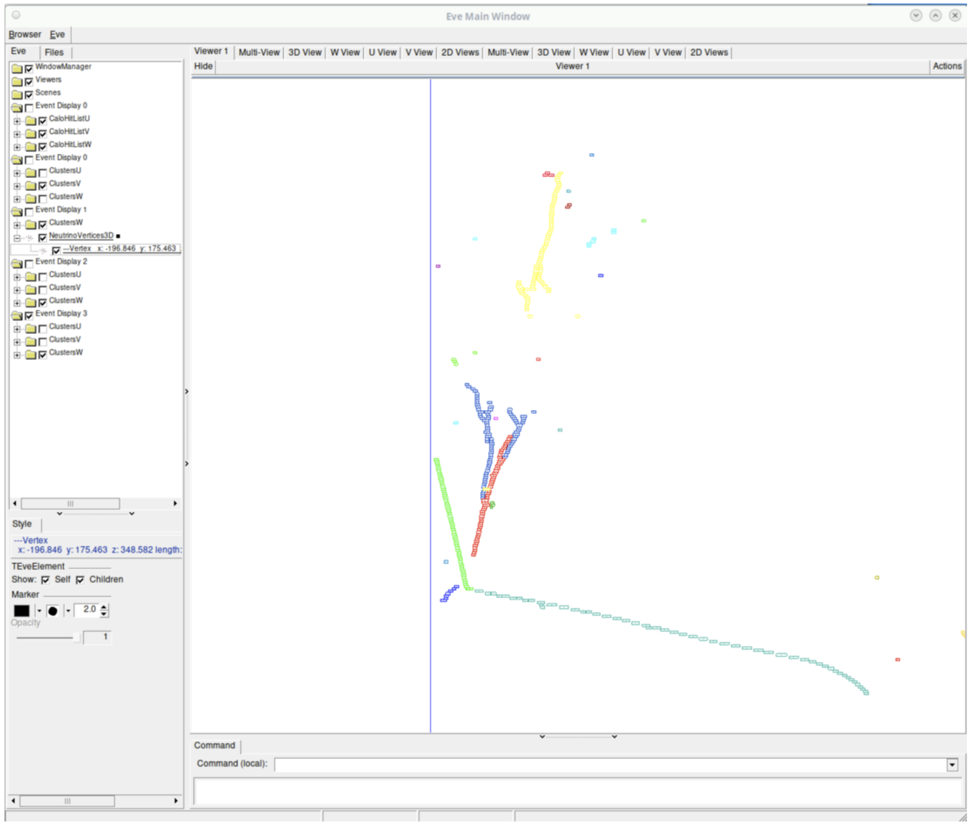
```
<!-- ThreeDShowerAlgorithms -->
...
<algorithm type = "LArThreeDShowers">
    <InputClusterListNameU>ClustersU</InputClusterListNameU>
    <InputClusterListNameV>ClustersV</InputClusterListNameV>
    <InputClusterListNameW>ClustersW</InputClusterListNameW>
    <OutputPfoListName>ShowerParticles3D</OutputPfoListName>
    <ShowerTools>
        <tool type = "LArClearShowers"/>
        <tool type = "LArSplitShowers"/>
        <tool type = "LArSimpleShowers"/>
    </ShowerTools>
</algorithm>

<algorithm type = "LArVisualMonitoring">
    <ClusterListNames>ClustersU ClustersV ClustersW</ClusterListNames>
    <ShowDetector>true</ShowDetector>
</algorithm>
```

- Re-run Pandora to see this new event display

## Understanding the reconstruction (8)

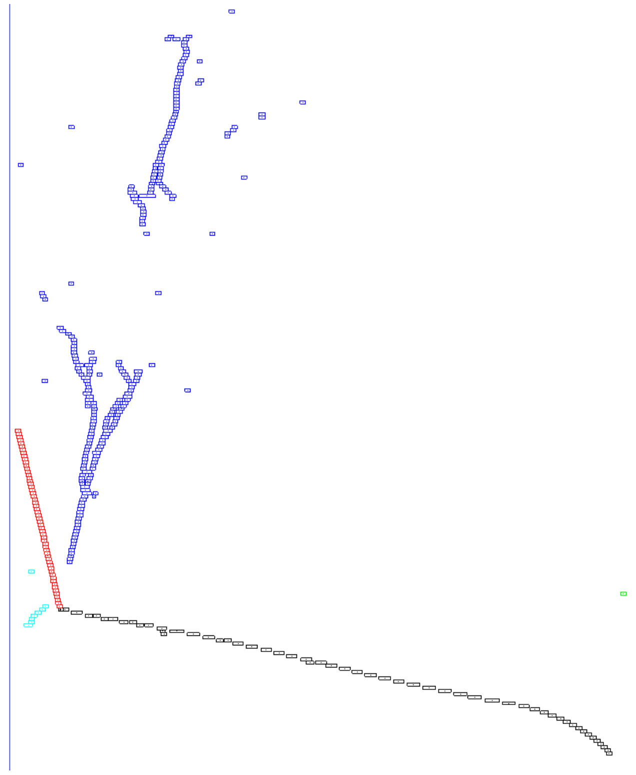
- Here we can see some merging of the shower clusters has been undertaken, but not as much as we'd like



## Fixing the reconstruction output

## The nature of the problem

- You’ve seen what the final event looks like, and you’ve seen how the reconstruction moves forward in a few key places
  - Broadly, the tracks are pretty good (except in the V view), but the showers are highly fragmented
- Now you need to see if you can modify the reconstruction algorithm chain to improve the outcome to something closer to the image on the right
- Choose from among the suggestions on the “pointers” slides in the previous section and the guide to those algorithms in the next section to try to improve the reconstruction
  - You may need to undertake some tuning of algorithm parameters for the best results – but don’t over-tune!
- If you make good progress on this event, feel free to explore other events in this file and fix those too!



## A very brief guide to adding algorithms to the XML (1)

- Let’s say you want to add the HitWidthClusterMerging algorithm to the end of the 2D clustering (we’ll look at just one of the views here, but remember the initial clustering runs for each view)
- The simplest addition is to add a single line which includes an algorithm with type LArHitWidthClusterMerging, that uses default parameters

```
<!-- TwoDReconstruction -->
...
<algorithm type = "LArClusteringParent">
    <algorithm type = "LArTrackClusterCreation" description = "ClusterFormation"/>
    <InputCaloHitListName>CaloHitListW</InputCaloHitListName>
    ...
</algorithm>
...
<algorithm type = "LArTransverseExtension"/>
<algorithm type = "LArHitWidthClusterMerging"/>
```

  - Note that while the class is called HitWidthClusterMergingAlgorithm, the algorithm name is LArHitWidthClusterMerging
  - This is a common mapping of class to algorithm name, but you can find the full set of mappings here, where you can search for the class name to find the corresponding XML algorithm type.
- If you want to modify parameters of the algorithm the update is slightly different…

## A very brief guide to adding algorithms to the XML (2)

- Let's say you want to add the HitWidthClusterMerging algorithm, but this time with a custom value for MinClusterSparseness
- Here the update now looks like this

```
<!-- TwoDReconstruction -->
...
<algorithm type = "LArClusteringParent">
    <algorithm type = "LArTrackClusterCreation" description = "ClusterFormation"/>
    <InputCaloHitListName>CaloHitListW</InputCaloHitListName>
    ...
</algorithm>
...
<algorithm type = "LArTransverseExtension"/>
<algorithm type = "LArHitWidthClusterMerging">
    <MinClusterSparseness>0.2</MinClusterSparseness>
</algorithm>
```

Note, the closing forward slash is dropped when we expand the block

  - Any parameters you want to modify are contained within the algorithm block
  - As will be noted in the next section, the collection of modifiable parameters can be found in each algorithm's ReadSettings function

## A Brief Guide to the Algorithms of Interest

## 2D clustering algorithms

## LArCrossGapsAssociation

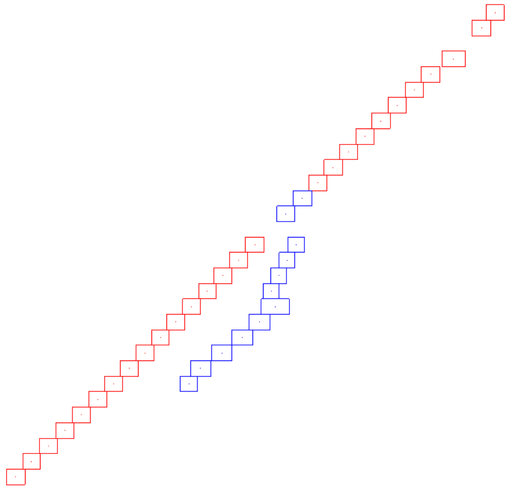
- This algorithm attempts to connect trajectories that may span gaps in the detector readout (e.g. dead channels, or gaps between APAs)
- As with many of the 2D algorithms, the clusters are first filtered according to selection criteria to determine if they should participate in the algorithm
  - A cluster must have a minimum number of hits and layers (you can think of a layer and a wire as synonymous)
  - The selection criteria and other parameters can be tuned in XML. The ReadSettings function is a useful resource
  - The resultant selected clusters are sorted by the starting layer
- Selected clusters are considered pairwise
  - Clusters are checked to see if they point in an approximately common direction and have start and end positions that could be consistent with belong to the same trajectory
  - Trajectories are then projected beyond a cluster in the direction of the target cluster and the target cluster is sampled to see if it is consistent with the projected trajectory
  - If enough points match (within a tolerance) the clusters can be associated
  - Clusters association must be bidirectional, it's not enough for one cluster to point to the other

## LArCrossGapsExtension

- This algorithm attempts to connect trajectories that may span gaps in the detector readout (e.g. dead channels, or gaps between APAs), taking a different approach to the previous algorithm
- Clusters are first filtered according to selection criteria
  - A cluster must have a minimum length
  - See the ReadSettings function for more tunable parameters
  - The resultant selected clusters are sorted by the number of hits
  - Clusters are then checked for proximity to a gap
- Selected clusters are considered pairwise
  - In this algorithm, a pointing cluster is created from each cluster, which considers the local trajectory more strongly than the overall cluster trajectory
  - If each pointing cluster is close enough to the position of the target pointing cluster across the gap and the angle across the gap is not too large, the clusters are associated
  - As before, associations must be bidirectional

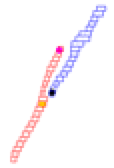
## LArOvershootSplitting

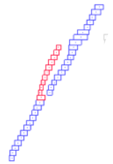
- This algorithm attempts to identify cases where hit clustering may have been overzealous, and continued to add hits where an alternative clustering was available
- Clusters are first filtered according to selection criteria
  - A cluster must have a minimum length
  - See the ReadSettings function for more tunable parameters
  - The resultant selected clusters are sorted by the number of hits
- Selected clusters are considered pairwise
  - The algorithm looks for locations where projections of one cluster intersect the other
  - If these distance between the clusters is not too large, the intersection of the projections is determined
  - If this intersection is sufficiently far along a cluster (i.e. we're not too close to a possible, legitimate, vertex), the cluster is considered for splitting



## LArBranchSplitting

- This algorithm attempts to identify instances where the continuation and branching of two clusters may have been wrong
- Clusters are first filtered according to selection criteria
  - A cluster must have a minimum length
  - See the ReadSettings function for more tunable parameters
- Selected clusters are considered pairwise
  - The algorithm looks for locations where one cluster appears to emerge from another
  - The angle shouldn't be too big (i.e. the clusters propagate in similar directions
  - If a redistribution of the hits leads to more consistent trajectories, the clusters are split





## LArKinkSplitting

- As the name suggests, this algorithm attempts to identify sharp direction changes within a cluster that may indicate a need to split the cluster
- Clusters are first filtered according to selection criteria
  - A cluster must have a minimum length
  - See the ReadSettings function for more tunable parameters
- Selected clusters are considered individually
  - The algorithm performs sliding fits along the cluster trajectory
  - The angle between fits that cover adjacent sliding windows is determined
  - The maximum angle across all fit comparisons is identified
  - If the angle is too large, the cluster is split at the midpoint of relevant fits

## LArTrackConsolidation

- This algorithm attempts to identify instances where shower-like clusters have stolen hits from track-like clusters
- Clusters are first filtered according to selection criteria
  - A track cluster must have a minimum length
  - A shower cluster must have a maximum length
  - See the ReadSettings function for more tunable parameters
- Selected clusters are considered pairwise
  - The algorithm checks to ensure the shower-like cluster is at most half the size of the track-like cluster
  - The algorithm then looks for shower hits that appear to fill gaps in the track-like trajectory
  - If such hits are found, they are removed from the shower-like cluster and added to the track-like cluster

## LArHitWidthClusterMerging

- This algorithm attempts to merge clusters involving wide hits
  - Wide hits are liable to occur when many drift electrons arrive at a small number of wires in quick succession
  - This produces a very wide signal from which a small number of hits are created
  - Such hits can be difficult to merge because the infinitesimal position of hits can be widely separated
- Clusters are first filtered according to selection criteria
  - A cluster must have a sufficiently large average hit width
  - See the ReadSettings function for more tunable parameters
- Selected clusters are considered pairwise
  - The algorithm looks for a potential merge point for the clusters (not necessarily end-to-end as wide hits can lead to overlap)
  - The individual clusters are checked for a sufficiently similar direction
  - The direction of a potential merged cluster is also compared to those original directions to ensure it does not change too much
  - If everything is consistent, the clusters are merged

## LArBoundedClusterMopUp

- This algorithm attempts to grow (by default) shower clusters
- Clusters are first filtered according to selection criteria
  - A cluster must have a minimum length
  - See the ReadSettings* function for more tunable parameters
- Selected clusters are considered pairwise
  - The algorithm finds particles produced by inter-plane matching and extracts their clusters (particle clusters)
  - The algorithm identifies clusters that did not produce particles by inter-plane matching (remnant clusters)
  - The edges of the shower are identified for the particle clusters by constructing bins along a track-like fit of the shower that span the perpendicular extent of the shower at that point
  - If a sufficiently large fraction of the hits in remnant clusters are bounded by the particle cluster edges, the clusters are merged

* Most algorithms in Pandora derive directly from the Algorithm class, but some, like this one, inherit from other classes. In this case, it can be useful to check if this base class has its own parameters (which you can then override in this algorithm's XML parameters), which are set in its own ReadSettings function. To see what extra parameters are available, you can look at its corresponding header file and see which class it inherits from. You can find the location of the corresponding base class by searching for the class in the LArContent class includes list.

## LArConeClusterMopUp

- This algorithm attempts to grow (by default) shower clusters
  - See the ReadSettings function for more tunable parameters
- Selected clusters are considered pairwise
  - The algorithm finds particles produced by inter-plane matching and extracts their clusters (particle clusters)
  - The algorithm identifies clusters that did not produce particles by inter-plane matching (remnant clusters)
  - A fit is performed through the hits in the particle cluster
  - Walking along that fit, the positive maximally transverse coordinate are collected and a fit performed
  - Walking along that fit, the negative maximally transverse coordinate are collected and a fit performed
  - These two “edges” about the overall fit define a search cone
  - If a sufficiently large fraction of the hits in remnant clusters are bounded by the particle cluster edges, the clusters are merged

## LArNearbyClusterMopUp

- This algorithm attempts to grow (by default) shower clusters
  - See the ReadSettings function for more tunable parameters
- Selected clusters are considered pairwise
  - The algorithm finds particles produced by inter-plane matching and extracts their clusters (particle clusters)
  - The algorithm identifies clusters that did not produce particles by inter-plane matching (remnant clusters)
  - The remnant clusters must have a minimum number of hits
  - If a remnant cluster is sufficiently close to a particle cluster, it can be merged

## 3D track algorithms

## LArThreeDTrackFragments

- This algorithm attempts to create particles in instances in which we have two well-reconstructed views, and a third fragmented view
  - See the ReadSettings function for more tunable parameters

- The algorithm ONLY considers clusters that are not owned by an existing particle
- It identifies pairs of clusters (in different views) with significant overlap in the drift coordinate (and thus likely to belong to the same particle)
- Starting from the ‘best pair’:
  - The clusters are used to create projected positions in the ‘missing’ view
  - For each projected position, the closest ‘available’ hit is sought
  - The clusters to which these hits belonged are together postulated to for the third cluster of the particle
  - If the, now, three-cluster match presents a consistent 3D image a particle is created

## 3D recovery algorithms

## LArVertexBasedPfoRecovery

- This algorithm attempts to recover any ‘tiny’ particles coming out of the neutrino vertex e.g. tiny proton stubs
  - See the ReadSettings function for more tunable parameters
- It first identifies all ‘available’ clusters ‘close-to’ the neutrino vertex
- It then attempts to ‘match’ found clusters across the views – to do this we refer to the overlap of the clusters in the drift coordinate and a ‘pseudo chi-squared’ metric that measures the extent to which a consistent 3D image can be formed
- First, three cluster matches are sought and particles are created in order of the best ‘chi-squared’
- Next, two cluster matches are sought, and again particle creation is prioritised by the ‘chi-squared’ metric

## LArParticleRecovery

- This algorithm attempts to reconstructed hitherto ‘missed’ particles
    - See the ReadSettings function for more tunable parameters
- First, it identifies all ‘available’ clusters above a user defined ‘size’
- If the algorithm is running in ‘vertex mode’, tiny stub-like clusters originating from clusters coming out of the vertex are also identified
- Pairs of matches across the three views are sought (a cluster can exist in multiple pairs) – to do this we refer to the overlap of the clusters in the drift coordinate
- These pairs are then examined to identify groups of clusters that exist within the same drift region
- If the group corresponds to a triplet or doublet of clusters (each belonging to a different view) a particle is created

## 3D mop up algorithms

## LArSlidingConePfoMopUp

- This algorithm attempts to optimise shower completeness by merging shower fragments into their parent shower
  - See the ReadSettings function for more tunable parameters
- First, all 3D showers and ‘small’ tracks are identified (these are the only particles considered in the algorithm)
- For each ‘large’ 3D shower:
  - we walk along it’s central axis, drawing a 3D cone at each point with an opening angle and length that increases with distance along the shower
  - any particles that are ‘significantly’ contained within a cone are recorded, alongside their potential parent
- After this process, a particle may have multiple potential parents, and ambiguities are resolved by picking the parent that most contained the particle
- Merges are then performed, and recursive merging is allowed (if running in vertex mode, which is the default, particles in ‘close’ proximity to the neutrino vertex cannot be merged)
- This whole process runs recursively, until no more merges can be made

## LArSlidingConeClusterMopUp

- This algorithm attempts to optimise shower completeness by merging 'lost' 2D clusters into their parent shower
    - See the ReadSettings function for more tunable parameters
- 'Significant' 3D showers and 'small' 3D tracks are collected and are the seed particles the algorithm will look to grow
- 'Small' 2D clusters, that do not yet belong to any particles, are identified and are the clusters the algorithm will use to grow the showers with
- For each seed particle:
    - we walk along it's central axis, drawing a 3D cone at each point
    - the cone is projected in each 2D view, and any clusters that are 'significantly' contained within a cone are recorded, alongside their potential parent
- After this process, a particle may have multiple potential parents, and ambiguities are resolved by picking the parent that most contained the particle
- Merges are then performed, and recursive merging is allowed

## LArIsolatedClusterMopUp

- This algorithm is one of the final algorithms to run in Pandora
- Its job is to make sure that as many of the input hits as is possible are included in the reconstruction output

  ⇒ limiting the missing energy in downstream energy estimations

  - See the ReadSettings function for more tunable parameters
- It first finds the 'seed' 3D particles to grow – this is configurable and is either all shower particles (default) or all showers and tracks
- It then identifies all remaining 'small' 2D clusters and 'dissolves' them into their 2D hits
- Each, now free, 2D hit is added to the closest cluster – **only if** the cluster is within some proximity
- The hits are added as 'isolated hits'
- Isolated hits **do not** contribute to trajectory fits but **do** contribute to energy calculations