Algorithm development



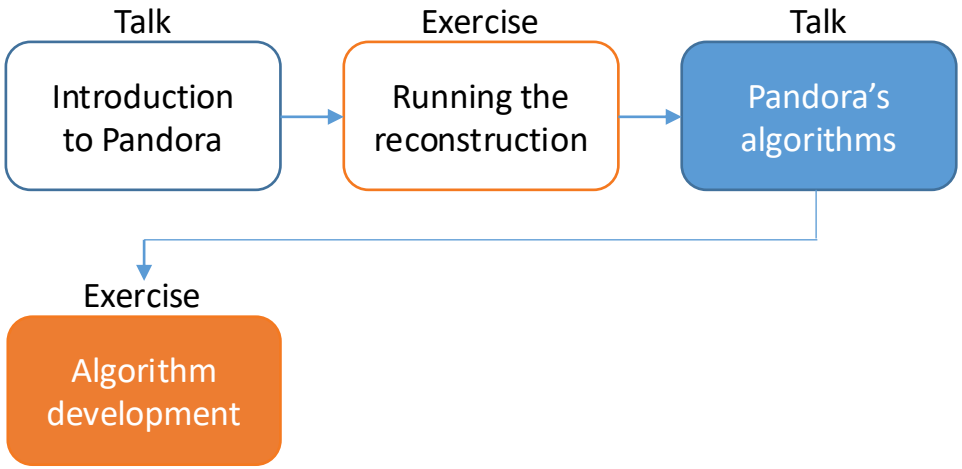
PANDORA PF

Andy Chappell and Alex Wilkinson for the Pandora team

October 2026

11th UK LArTPC Software and Analysis Workshop

Reconstruction session



Key references:

[Pandora ProtoDUNE paper]
[Pandora MicroBooNE paper]
[Pandora Vertexing paper]

Workflows

Depending on the experimental context, Pandora can provide a number of different reconstruction workflows. For example, in DUNE:

- Long baseline workflows (HD and VD)
- Atmospheric workflow (HD, VD pending)
- Low energy (HD, VD)
- ProtoDUNE (SP, DP, HD, VD)

There workflows are tailored to meet the specific needs of the use case

- Typically, much in common, but with unique details, such as handling of isotropic events in atmospherics, cosmic rays in SBND, etc

Some use cases need more specific treatment

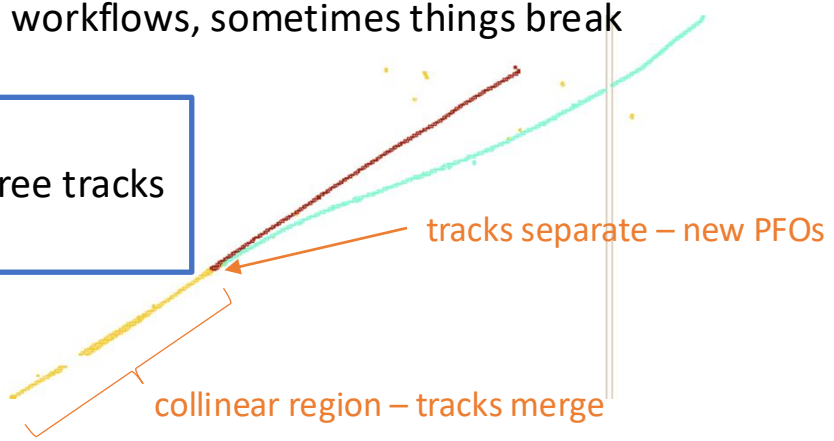
- Long baseline (GeV scale) workflow is not appropriate for a Supernova neutrino sample

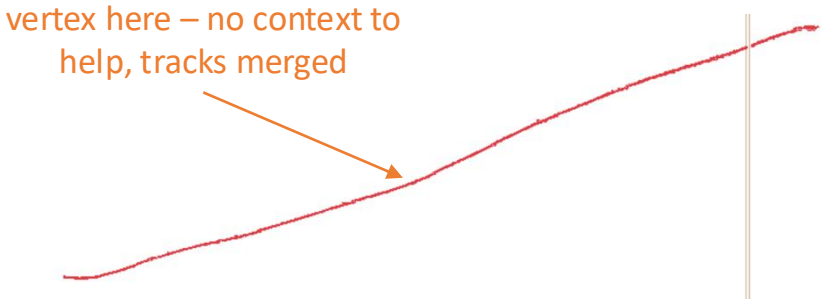
When things go wrong

Reconstructing neutrino interactions can be extremely challenging

Whether its new samples or well-established workflows, sometimes things break

Example:
Initial hit sharing leads to three tracks where there should be one

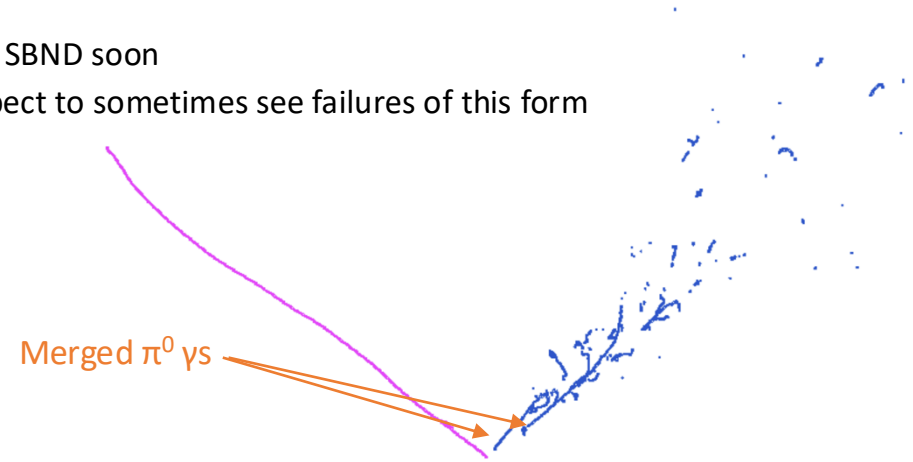




Example:
Back-to-back trajectories can be merged. This can be recoverable if the tracks are calorimetrically separable

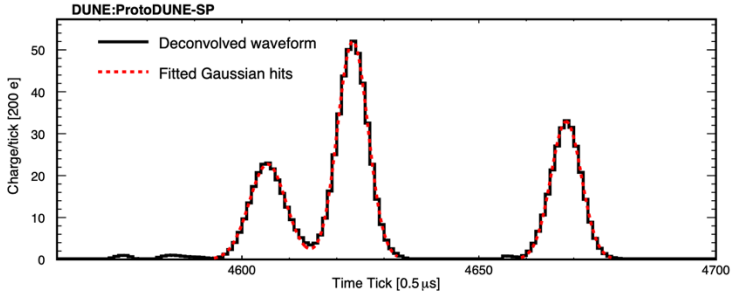
When things go wrong

- For showers, in particular, a small opening angle between two showers can make it challenging to determine to which shower the hits belong
- This can result in an incorrect distribution of hits among the showers, or to a complete merging of the two showers
- In DUNE, we have a new transformer network targeting shower clustering that can handle many such case
  - We plan to port this to SBND soon
  - But you should still expect to sometimes see failures of this form

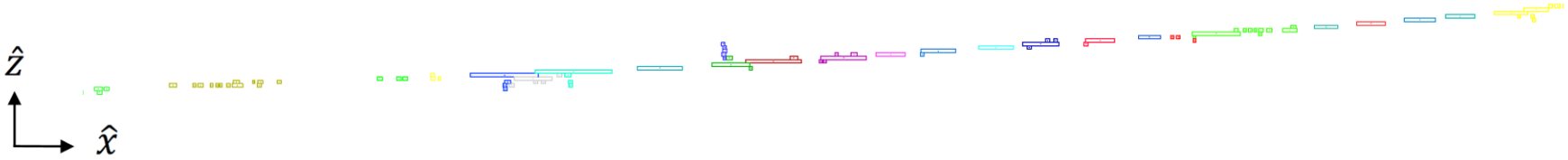


When things go wrong

- Pandora runs on discrete hits that it receives from signal processing steps that run before it
- However, those hits are extracted from continuous waveforms produced by the drift electrons as they pass by induction planes and get deposited on the collection plane



- If a particle trajectory causes its drift electrons to interact with a single wire/strip, the waveform can become very wide, making it challenging to extract individual hits
  - The result is **a small number of wide hits** (i.e. a high uncertainty in the position along the drift direction)
  - Such hits can be difficult to cluster



When you hit problems with your analysis

- Understanding the reconstruction can really help refine your analysis work
- A classic example is constructing a selection for your analysis, and finding purity or efficiency is not where you want it to be. What should you do?
- Step 1: Look at your events!
  - You'll learn things. There may be features of your sample you hadn't considered
  - Some events just won't be recoverable, you can often identify these by eye rather quickly
  - Identify a handful of events that pass your selection, and a handful that don't
  - It doesn't matter too much which EVD you use, just be sure to use one
  - Do the events look well reconstructed?
- Step 2: Is there a common theme to any selection errors?
  - If so, do they appear to be induced by reconstruction errors?
- Step 3: Talk to the Pandora team
  - If you think reconstruction might be the cause of any problems, please contact us
  - It's useful if you can tell us which workflow you're running, on what sample, and ideally provide a few illustrative cases of the failures
- Step 4: Fix it…

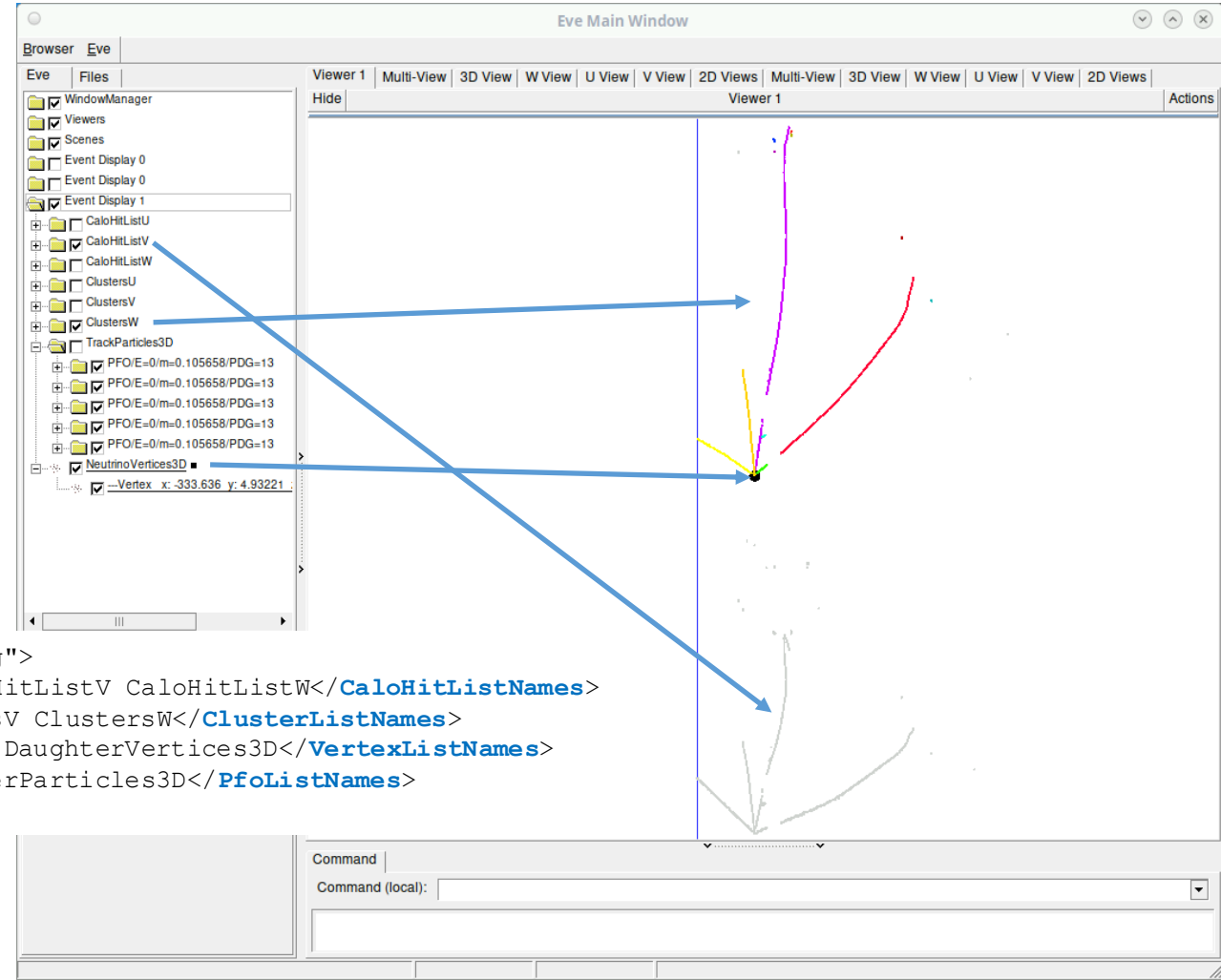
Fixing problems

- Sometimes step 4 can simply be an issue of usage, or configuration, so we can point you in the right direction
- Sometimes there will be a real reconstruction failure to fix and the Pandora team can look to address it, or…
- You can develop an algorithm yourself
  - Pandora isn't a closed shop
  - We do have full code review processes, coding style, etc, but anyone can submit a new algorithm
  - If you do want to pursue this route, we recommend contacting us early

- The rest of this tutorial will give a brief introduction to developing clustering algorithms for Pandora

Visualisation

- Pandora allows you to look at reconstruction status at any point in the workflow
- Below is the XML code to produce this display
- Which elements are available will depend on where you are in the reconstruction




```
<algorithm type = "LArVisualMonitoring">
  <CaloHitListNames>CaloHitListU CaloHitListV CaloHitListW</CaloHitListNames>
  <ClusterListNames>ClustersU ClustersV ClustersW</ClusterListNames>
  <VertexListNames>NeutrinoVertices3D DaughterVertices3D</VertexListNames>
  <PfoListNames>TrackParticles3D ShowerParticles3D</PfoListNames>
</algorithm>
```

- You can also produce in-algorithm displays

Visualising ground truth

- To help you assess reconstruction, I've created a feature branch that provides ground truth information
- You can access this feature by checking out a feature branch from Pandora's main repository (**hopefully, you did this on Monday**)

To only build Pandora:
ninja -C $MRB_BUILDDIR larpandoracontent/install -j 3

```
$ git remote add pandora git@github.com:PandoraPFAOrg/LArContent.git
$ git fetch pandora
$ git checkout pandora/uk_lartpc_workshop_2026
$ git switch -c uk_lartpc_workshop_2026          # hyphen character encoding varies, copy/paste might not work
```

- This feature will be made available in a future tagged release
- You can add ground truth display after the `LArPreProcessing` algorithm in the Master XML

```
<algorithm type = "LArGroundTruthMonitoring" />
```

- This will show you true clusters, PID, hierarchy tier, and neutrino vertex

This requires Pandora FHiCL parameter EnableMCParticles to be set to true

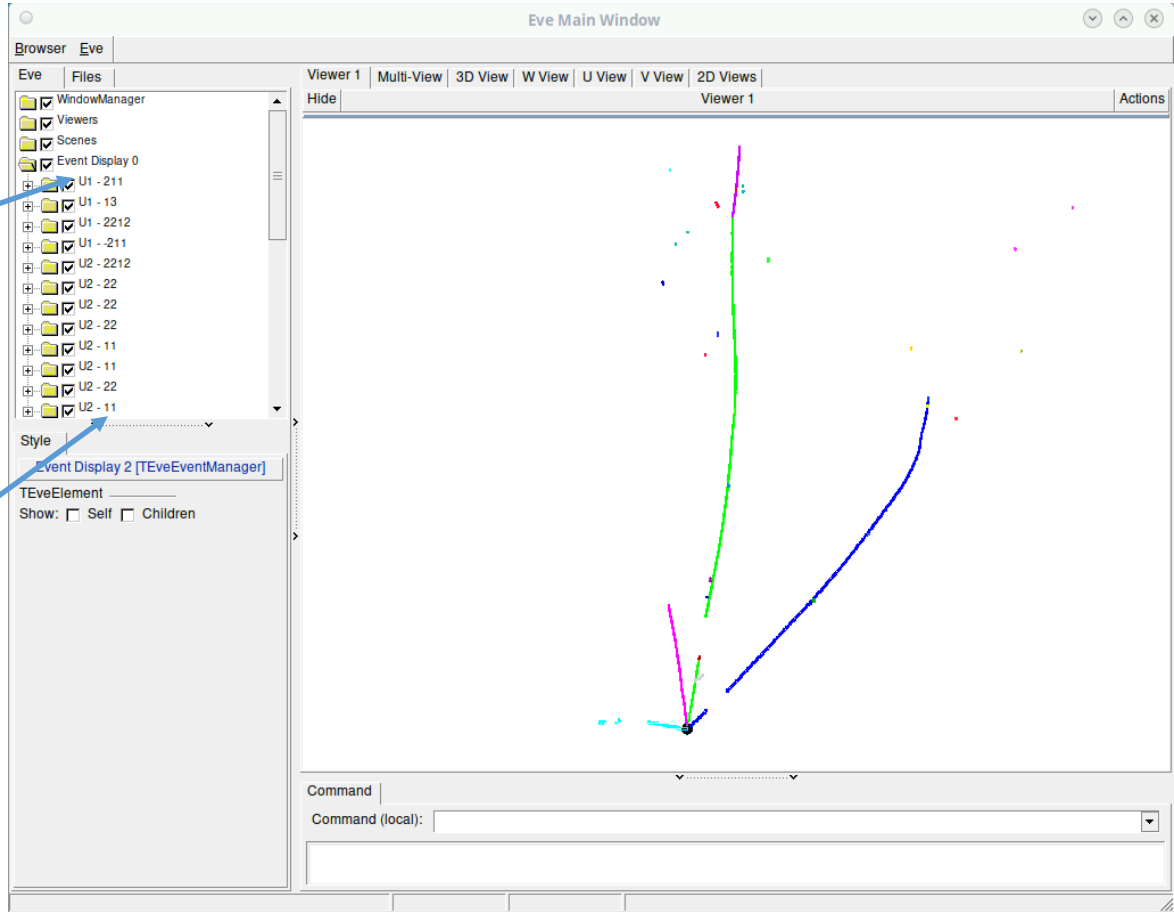
You can find an example XML containing this algorithm in
`$MRB_TOP/srcs/larpandoracontent/scripts/PandoraSettings_Master_SBND_Reduced.xml`
Note, Pandora configuration XMLs usually live in the `sbndcode` package

Visualising ground truth

- Here we have an example showing the U view

View and tier

PDG code



Creating clusters

For this session, you can use the root files you've already created during the TPC simulation tutorial. As noted previously, we'll be using the Pandora event display in this tutorial – this requires a custom FHiCL file to avoid some graphics library conflicts. Create a reco_driver.fcl file and populate it with:

This is a very convenient way to customize FHiCL – inherit and minimally modify

```
#include "standard_reco2_sbnd.fcl“

physics.reco2: [ rns, pandora ]
physics.end_paths: []

physics.producers.pandora.ConfigFile: "PandoraSettings_Master_SBND_Reduced.xml"
physics.producers.pandora.EnableMCParticle: true
```

Start from the standard reco chain

Only run Pandora pat. rec.
Produce no output

Point to our new XML settings file

Pass MCParticles to Pandora

Question: Is the path for this file in your FHICL_FILE_PATH?

Creating clusters

Let's make sure we can run Pandora and see an event display with our custom reco driver fcl before looking into algorithms.

First, make sure the fcl knows where to look for the Pandora configuration xml we are using:

```
$ export FW_SEARCH_PATH=${MRB_SOURCE}/larpandoracontent/scripts:${FW_SEARCH_PATH}
```

Now, run reco_driver.fcl over the reco1 ROOT file you made in the previous Pandora tutorial, you should see execution pause and an event display window appear. Hit enter in the terminal to continue execution to the next event display. Pro-tip: press "w" when in the event display to make it look not horrible.

You should also try adding the groud truth display in the Master xml just after the `LArPreProcessing` block:

```
<algorithm type = "LArGroundTruthMonitoring" />
```

Running reco_driver.fcl again, you should see the ground truth display

A template algorithm

- Given the time available for this session, I can't give a lengthy exposition on Pandora's SDK capabilities, so I'll provide a very basic template algorithm, covering some core features, that you can adapt
- We're going to look at 2D clustering
- You'll find this template as part of the feature branch you checked out earlier, in

  `larpandoracontent/LArTwoDReco/LArClusterCreation/TutorialClusteringAlgorithm.*`

The previously referenced XML also links to the template algorithm found in
`$MRB_TOP/srcs/larpandoracontent/scripts/PandoraSettings_Neutrino_SBND_Reduced.xml`

Lists and parameters

- Pandora operates on lists of objects, managed (and strictly controlled) by the SDK
- There are lists of hits, clusters, particle flow objects (PFOs in Pandora = PFPs in LArSoft), etc

```
StatusCode TutorialClusteringAlgorithm::Run()
{
    const CaloHitList *pCaloHitList{nullptr};
    PANDORA_RETURN_RESULT_IF(STATUS_CODE_SUCCESS, !=,
        PandoraContentApi::GetList(*this, m_caloHitListName, pCaloHitList));
    …
}

StatusCode TutorialClusteringAlgorithm::ReadSettings(const TiXmlHandle xmlHandle)
{
    PANDORA_RETURN_RESULT_IF(STATUS_CODE_SUCCESS, !=,
        XmlHelper::ReadValue(xmlHandle, "CaloHitListName", m_caloHitListName));
    return STATUS_CODE_SUCCESS;
}
```

The list into which we want to read hits

The name of the list we want to read

- `ReadSettings` is a standard function that is always called before any algorithm is run
  - You populate this with the XML tags you want to read
- `Run` is a standard function that is always called to execute the logic of an algorithm
  - You populate this with the core algorithm logic you want applied

In algorithm visualisation

- One very useful thing you can do inside an algorithm, is produce event displays of intermediate state

```
StatusCode TutorialClusteringAlgorithm::Run()
{
    …
    PANDORA_MONITORING_API(
        SetEveDisplayParameters(this->GetPandora(), false, DETECTOR_VIEW_XZ, -1, 1, 1));

    PANDORA_MONITORING_API(VisualizeCaloHits(this->GetPandora(), pCaloHitList, "Hits", GRAY));

    PANDORA_MONITORING_API(ViewEvent(this->GetPandora()));
    …
}
```

Initialise the event display (one time operation)

Any number of visualisation commands

Render the scene (once for each collection of visualisations you want to display)

- Here, we're simply showing the hits that have been read in, little different to the LArVisualMonitoring algorithm, but you can produce custom hit lists, lines, markers, etc

Creating clusters 1

- To create clusters, you need to construct a temporary list to store them and subsequently save that to a named list

```
StatusCode TutorialClusteringAlgorithm::Run()
{
    …
    const ClusterList *pTemporaryList(nullptr);
    std::string temporaryListName;
    PANDORA_RETURN_RESULT_IF(STATUS_CODE_SUCCESS, !=,
        PandoraContentApi::CreateTemporaryListAndSetCurrent(*this, pTemporaryList, temporaryListName));

    …

    if (!pTemporaryList->empty())
    {
        PANDORA_RETURN_RESULT_IF(STATUS_CODE_SUCCESS, !=,
        PandoraContentApi::SaveList<Cluster>(*this, m_outputClusterListName));
        PANDORA_RETURN_RESULT_IF(STATUS_CODE_SUCCESS, !=,
            PandoraContentApi::ReplaceCurrentList<Cluster>(*this, m_outputClusterListName));
    }
    …
}
```

Create a list and set it as the current list.

Each new cluster will be added to a temporary named list, recorded here

The location in memory where cluster objects will be held

Assign the clusters to a “permanent” named list

Make the new list the default cluster list (not strictly necessary)

Creating clusters 2

- To add hits to a cluster, you create a cluster with a seed hit, and then add hits to that cluster

```
StatusCode TutorialClusteringAlgorithm::Run()
{
    …
    const Cluster *pCluster(nullptr);
    for (const CaloHit *const pCaloHit : *pCaloHitList)
    {
        if (!PandoraContentApi::IsAvailable(*this, pCaloHit))
            continue;
        if (!pCluster || (rand() % 10) == 0)
        {
            PandoraContentApi::Cluster::Parameters parameters;
            parameters.m_caloHitList.emplace_back(pCaloHit);
            PANDORA_RETURN_RESULT_IF(STATUS_CODE_SUCCESS, !=,
                PandoraContentApi::Cluster::Create(*this, parameters, pCluster));
        }
        else
        {
            PANDORA_RETURN_RESULT_IF(STATUS_CODE_SUCCESS, !=,
                PandoraContentApi::AddToCluster(*this, pCluster, pCaloHit));
        }
    }
    …
}
```

Pandora's SDK does not permit hits in multiple clusters, so check availability

Start a new cluster at random

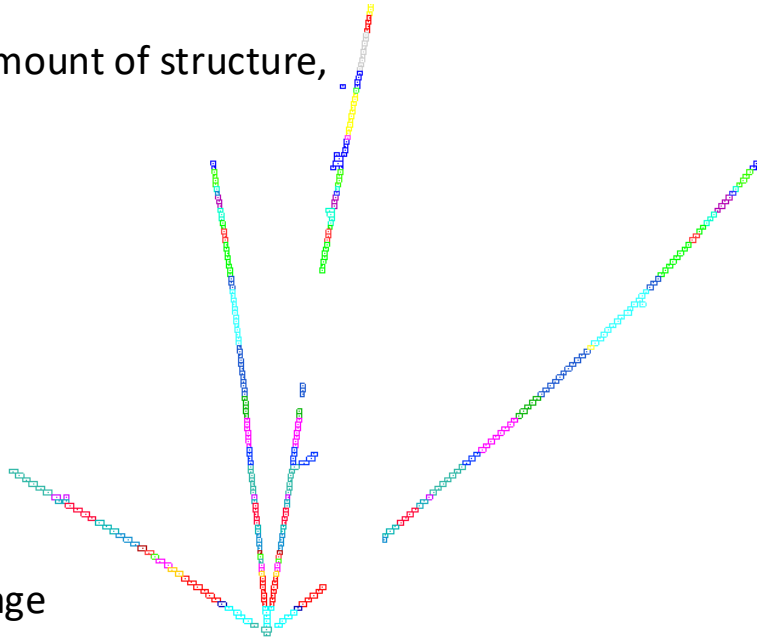
Add the seed hit

Create the cluster and receive the object

Continue with the current cluster and add the next hit

Creating clusters 3

- The example shown clusters hits randomly, creating a new cluster once every 10 hits on average
- In this case, the initial ordering of the hits leads to some amount of structure, but obviously it's not the greatest algorithm ever written
- Your task, try to create something with a bit more logic
- Potentially useful resources:
  - CaloHit
  - LArClusterHelper (mainly for hit sorting functions)
  - LArPcaHelper
  - LArTwoDSlidingFitResult
  - KalmanFilter
  - KDTreeLinkerAlgoT
- Grepping for some of these can provide some example usage
  - Suggest being selective about what you think you might want. You could easily spend the entire tutorial just reading the code
- Good luck!



Additional Task - Merging clusters

Merging clusters

- The previous example showed how to make new clusters from input hits
- Here, we'll look at how you can merge existing clusters together
- You'll see a lot of familiar structure in managing the lists
- You'll find this template as part of the feature branch you checked out earlier, in

```
larpandoracontent/LArTwoDReco/LArClusterAssociation/TutorialClusterMergingAlgorithm.*
```

The previously referenced XML also links to the template algorithm found in

```
$MRB_TOP/srcs/larpandoracontent/scripts/PandoraSettings_Neutrino_SBND_Reduced.xml
```

Cluster merging 1

- As before, we begin with some list processing to get our input clusters. It's possible that in this case, you may want to apply some "pre-selection" to filter the clusters you want to work with

```
StatusCode TutorialClusterMergingAlgorithm::Run()
{
    const ClusterList *pInputClusterList{nullptr};
    PANDORA_RETURN_RESULT_IF(STATUS_CODE_SUCCESS, !=,
        PandoraContentApi::GetList(*this, m_inputClusterListName, pInputClusterList));
    if (!pInputClusterList || pInputClusterList->empty())
        return STATUS_CODE_SUCCESS;
    ClusterVector selectedClusters(pInputClusterList->begin(), pInputClusterList->end());
    …
}
```

The list into which we want to read clusters

The name of the list we want to read

Copy our list to avoid iterator issues when deleting clusters

Cluster merging 2

- We can loop over the clusters to make decisions about merging
- First priority is to keep track of what you're doing to avoid attempting invalid merges (which Pandora will reject)

```
StatusCode TutorialClusterMergingAlgorithm::Run()
{
    …
    ClusterList deletedClusters;
    for (const Cluster *const pParentCluster : selectedClusters)
    {
        if (std::find(deletedClusters.begin(), deletedClusters.end(), pParentCluster) != deletedClusters.end())
            continue;
        …
    }

    return STATUS_CODE_SUCCESS;
}
```

Keep track of the clusters we're deleting

Outer loop over our selected clusters

Skip over anything we've already deleted

Cluster merging 3

- We can loop over the clusters to make decisions about merging
- Once you identify a merge, the API call `MergeAndDeleteClusters` handles the details

```
StatusCode TutorialClusterMergingAlgorithm::Run()
{
    …
    for (const Cluster *const pParentCluster : selectedClusters)
    {
        …
        for (const Cluster *const pChildCluster : selectedClusters)
        {
            if (!pChildCluster || pParentCluster == pChildCluster) continue;
            if (std::find(deletedClusters.begin(), deletedClusters.end(), pChildCluster) !=
                deletedClusters.end())
                continue;
            if (!this->AreClustersMergeable(pParentCluster, pChildCluster))
                continue;
            deletedClusters.push_back(pChildCluster);
            PANDORA_RETURN_RESULT_IF(STATUS_CODE_SUCCESS, !=,
                PandoraContentApi::MergeAndDeleteClusters(*this, pParentCluster, pChildCluster));
        }
    }
    …
}
```

Inner loop over our selected clusters

Skip over anything we've already deleted

Logic to determine merging decision

Track our deleted clusters

Make the merge and delete the child cluster

Cluster merging 4

- The core processing loop is short and simple
- The merging logic is passed to a separate function, which in this case, makes a random decision
- Your task is to replace this with something more reasonable

```
bool TutorialClusterMergingAlgorithm::AreClustersMergeable(
    [[maybe_unused]] const Cluster *const pParentCluster,
    [[maybe_unused]] const Cluster *const pChildCluster) const
{
    return (rand() % 10) == 0;
}
```

Annotation tells the compiler that unused variables are ok