The BlurredCluster Reconstruction Technique

Michael Wallbank[*1]

[1]University of Sheffield, Sheffield, UK

Contents

[*]email: m.wallbank@sheffield.ac.uk

1 Introduction

This document describes the Blurred Clustering algorithm for use forming initial 2D clusters within the LArSoft framework. The method specialises in shower reconstruction and has been developed for DUNE (specifically for the 35-ton prototype); it is however a flexible algorithm so is straight forward to tune for other LAr experiments (and has already been in, e.g., LArIAT).

In LArTPCs, 2D reconstruction ('clustering') is performed for each TPC plane in the space with wire number on the x-axis and time on the y-axis. This defines a 2D space with axes representing dimensions along an APA face and normal to it. The clustering takes hits as input, found by a LArSoft hit finder such as `gaushit`.

The separate algorithms, BlurredClusteringAlg, TrackShowerSeparationAlg and MergeClustersAlg, are controlled by an art producer module, BlurredClustering_module.cc, which lives in larreco/ClusterFinder and places the output clusters and associated hits in the event record. This is overviewed briefly in section 2.

The BlurredCluster technique is unique in that before performing any clustering it applies a Gaussian smearing to the particle hits in order to form fuller and more complete clusters. The implementation of the algorithm is written in the BlurredClusteringAlg class within larreco/RecoAlg, and the methods employed are discussed in detail in section 3. Other algorithms have also been designed to various levels of completeness to improve the reconstruction in different ways. These include cluster merging, written in the MergeClusterAlg class, which runs on the output of the clustering and merge smaller clusters together which are typically fragmented by the blurring method, and a track shower separation algorithm, written in TrackShowerSeparationAlg, designed to filter out hits which were potentially from tracks rather than showers. These is written very generically and presented briefly in section 4. A general discussion on configuring the clustering and the reconstruction performance are contained in section 5 and section 6 respectively.

2 The Reconstruction Module

An art producer module controls the running of the algorithms and is configured as usual within a fcl steering file, using `@local_blurredcluster`. There are a few user-defined parameters for configuring the overall running of the reconstruction, described in the following subsections.

The module takes the hits from the event (specified by `HitsModuleLabel`) and performs the reconstruction in each relevant space (see section 2.2) by calling BlurredClusteringAlg (and any further algorithms, as specified) on the selection of hits. The output clusters are placed into the event record for use in downstream reconstruction and analyses.

2.1 User-Defined Parameters

The configurable parameters relevant to the running of the art module, along with their default values, are as follows:

- `bool CreateDebugPDF` (false) – whether or not to create a demonstratory document to show the reconstruction process (see section 2.3).
- `bool MergeClusters` (false) – whether or not to run the cluster merging algorithm over the initial clusters (see section 2.4).
- `bool GlobalTPCRecon` (true) – whether or not to apply the reconstruction in a global TPC space (see section 2.2).
- `bool ShowerReconOnly` (false) – whether or not to initially run the track shower separation algorithm to attempt to separate the hits (see section 2.4).

2.2 Global TPC Recon

There are two ways in which the reconstruction can be performed in a given detector: either for each cryostat, TPC and plane separately or alternatively across all TPCs for each plane in a given cryostat. The latter involves defining some kind of 'global wire' coordinate which apply across multiple TPCs for a given plane; this concept is demonstrated in Fig. 2.1. The Boolean parameter `GlobalTPCRecon` is the switch to control which of these methods is applied.

This scheme allows reconstruction to take place in a more natural way: in a single large volume without being broken up by arbitrary TPC boundaries. Note the figure shows this convention for a single plane but it is a trivial extension to each plane in turn. This is applied for each drift volume separately, with showers passing through APA planes requiring splitting at this 2D stage as there is no way to match this up without 3D reconstruction.

2.3 Create a Debug PDF

The Boolean variable `CreateDebugPDF` determines whether or not a *.pdf* file is created showing the blurring process in its various stages. As the name suggests, it is particularly useful for debugging since it shows the separate stages of the reconstruction process and the effect of each on the clustering. The file contains a page for each TPC/plane combination for every event, each showing four images demonstrating the method (see an example in Fig. 2.2). Turning on the creation of this debugging resource can prove incredibly useful in determining if the clustering is performing as expected but there are a couple of things to be considered:

- the document produced has $n_{cryostats} \times n_{TPCs} \times n_{planes} \times n_{events}$ pages and so is a significant size – potentially on the order of gigabytes if many events are reconstructed;

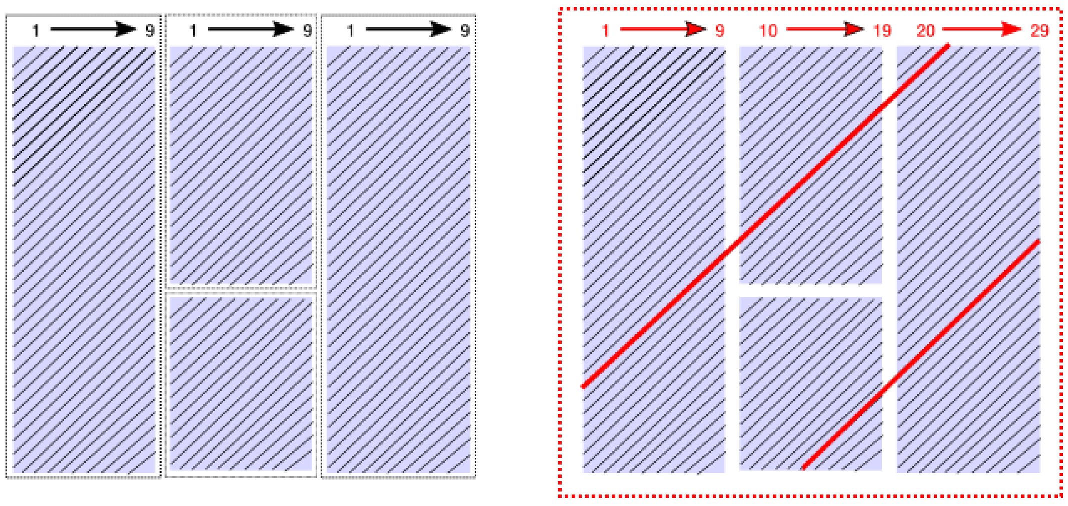


Figure 2.1: Demonstration of the concept of a 'global wire' number. The blue rectangles represent APAs in a LArTPC (with layout and dimensions similar to that of the 35-ton prototype), along with hypothetical wires and wire numbers represented in black. In the left-hand plot the conventional numbering system is demonstrated, with numbering restricted to each TPC separately. Each wire is thus represented by four numbers: the cryostat, the TPC, the plane and the wire numbers. In the right-hand plot, the scheme of labelling wires globally is shown. An imaginary line connecting wires which would overlap others on different APAs (and thus in different TPCs) if extended is used to give each of these wires a common index. In this system, each wire is thus described using just three numbers: cryostat, plane and wire.

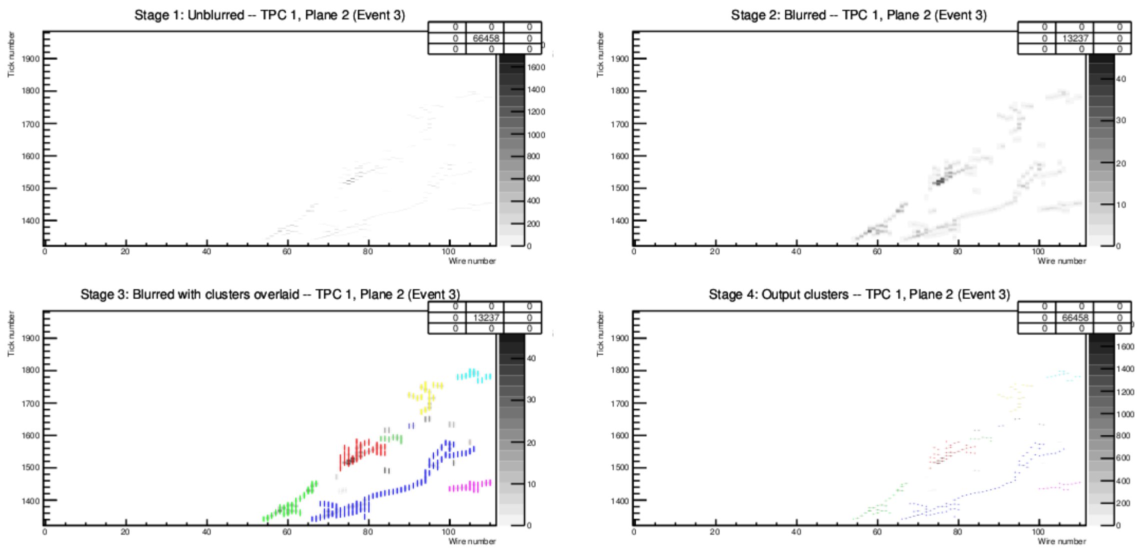


Figure 2.2: Example page from a debug pdf created by the BlurredCluster module when the `CreateDebugPDF` parameter is set to `true`.

- producing this file during the reconstruction is much more computationally expensive and so slows down running of the module by around a factor of four.

The `CreateDebugPDF` parameter should therefore be set to `false` during normal running.

2.4 Further Parameters

As briefly indicated earlier, there are other algorithms which are written to work with the 2D reconstruction. These are the MergeCluster algorithm and the TrackShowerSeparation algorithm. These are in general outdated and written before many BlurredCluster updates and are thus turned off by default. With 3D reconstruction development ongoing, the code will remain in LArSoft until such a time as they are no longer required; until then their use is available but discouraged.

The methods are motivated and described in section 4.

3 Blurred Cluster Technique

This section discusses the technique of Blurred Clustering and its various methods in the context of its implementation within LArSoft. The history of the method is outlined briefly in section 3.2 before the algorithm is detailed in section 3.3. User defined parameters are described in section 3.4.

3.1 Overview

The method of Blurred Clustering uses a Gaussian kernel to smear the hit map and introduce fake hits, creating a more isotropic distribution and allowing more complete clusters to be formed. The Gaussian is convolved with an image of the hit map to achieve this before clustering proceeds on the blurred hits, where nearby hits over a certain charge threshold are clustered together.

3.2 History of the Algorithm

The technique of using a Gaussian kernel to introduce blurring to an image is widely used in image processing [1]. The original implementation of the algorithm as used here was written by Asher Kaboth (Imperial College London) for use in the DMTPC experiment and is described in his PhD thesis [2]. This algorithm was subsequently rewritten for use in the ECal reconstruction for ND280, the T2K near detector, by Ben Smith (University of British Columbia) and was developed further in ND280 and within LArSoft by the author.

The version of the code currently within LArSoft has practically been completely rewritten following the work of the aforementioned people; it would be improper not to ignore their contributions however and so their previous work is gratefully acknowledged.

3.3 Algorithm Details

The Blurred Clustering algorithm comprises many different parts which will be discussed in detail in this present section. fhicl parameters read in at run-time which can be altered will be mentioned in `typewriter` and then listed for reference in section 3.4.

3.3.1 ConvertRecobHitsToVector

ConvertRecobHitsToVector is the first method called and takes a vector of recob::Hits as input. Two data objects are filled in this function, both two dimensional, fixed size vectors. The vectors are filled once per hit and represent the hit map received by the function, the indices indicating the wire and tick position of the hit.

The main object used throughout the implementation of the algorithm contains just the charge deposited by each hit, indexed by position as described. This vector is used to perform the blurring and clustering at a later stage. The second object stores an `art::Ptr<recob::Hit>` object for each true hit, before any reconstruction takes place. This is used latterly to convert the output clusters from position back to hit.

3.3.2 FindBlurringParameters & GaussianBlur

The main functionality of the blurring technique is written into these methods. The final output is another 2D vector containing the blurred image of the hit map returned by **ConvertRecob-HitsToVector** after it has been smeared by convolution with the Gaussian kernel.

The kernel used by the algorithm takes the form

$$\frac{1}{\sqrt{2\pi}\sigma_{wire}}\frac{1}{\sqrt{2\pi}\sigma_{tick}}e^{-\frac{r_{wire}^2}{2\sigma_{wire}^2}}e^{-\frac{r_{tick}^2}{2\sigma_{tick}^2}}, \tag{1}$$

and has four variables: the standard deviation σ of the Gaussian function in each dimension and the 'blurring radius' r, the distance to extend the blurring in each direction. An example of such a kernel is demonstrated in Fig. 3.1.

FindBlurringParameters runs first and sets the values in the kernel expression. It attempts to do this in such a way so as to follow the trajectory of the particle as closely as possible, so as to encompass the entire track or shower, including all its fragmented parts, in a big blurry blob. The effectiveness of the clustering is directly related to the accuracy with which this stage can be accomplished; the correct blurring can resolve even closely parallel showers. In order to apply the most appropriate blurring possible to the plane, a Principal Component Analysis (PCA) is used to find an initial estimate of the directionality of the particles. The unit vector found from the PCA is used to scale four user defined parameters, `BlurWire` and `BlurTick`, `SigmaTick` and `SigmaWire`, accordingly so the final Gaussian shape and blurring radii are dependent on some preliminary hypothesis about the tracks/showers along with tunable parameters which can be used to adjust the clustering as necessary. For an example of how a scaled 2D kernel can look, refer back to Fig. 3.1.

It should be noted that these parameters are again scaled upon applying the convolution with the hit map based on the width (in time) of each hit individually. In order to save on

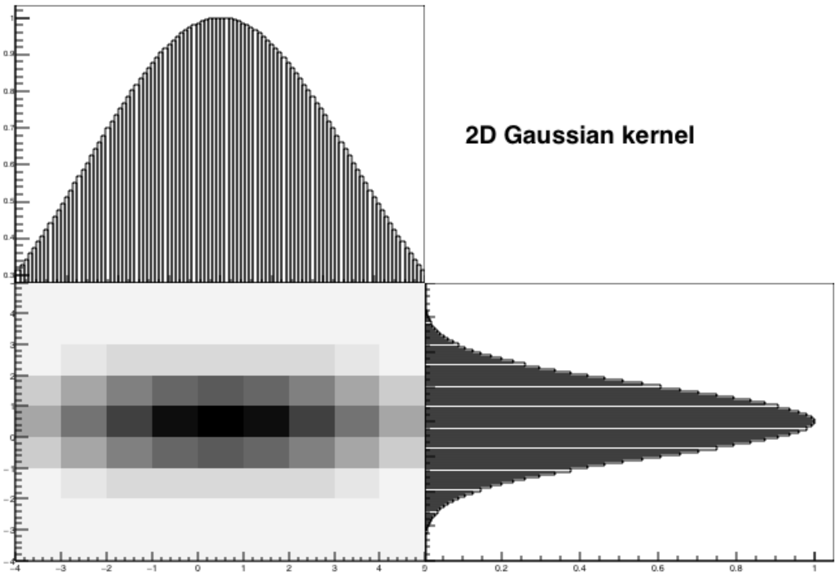


Figure 3.1: An example of a 2D kernel produced from Eq. 1. To visualise the blurring process, consider the charge at point (0, 0) to be a hit. The positions around it, within a given blurring radius r (in this case 4 in each dimension) are assigned charge according the value of the kernel at this point, thus introducing fake hits. In this example, the sigma in the wire direction is much greater so one can imagine this representing hits which are associated to a shower travelling preferentially along the wire direction.

computation, all possible kernels (Eq. 1) that can be required (determined up to some upper-defined parameter `MaxTickWithBlur`) are calculated and stored in a member function upon construction of the class.

The **GaussianBlur** method is then called to apply the smearing of the hit map. It proceeds by looping over all hits in the blurring region and spreading out the charge in the 2D region around it as determined by the kernel. The kernel used is unique for each hit, in order to achieve the most accurate blurring possible, and is determined by the parameters found from the **FindBlurringParameters** function but also from the width of each hit in time. The 'hit' concept is a useful one in LAr reconstruction but just considering a hit as a 2D point in wire/tick space is an incomplete representation. Drift electrons arriving at the wire from different places and in different directions leave very different signatures on the wires, a hugely important one being its 'width'. The width of each hit, taken as the RMS of the fitted hit Gaussian, is used to scale the width of the blurring Gaussian in the tick direction by adding it to the scaled SigmaTick parameter in quadrature.

This method introduces so-called 'fake hits' which are used when applying the clustering methods. It is these that allow the more complete clusters to be formed, by filling in the gaps in particle deposits and assisting with reconstruction of sparse hits, and are removed before the final clusters are output.

The output blurred hit vector is then analysed by **FindClusters** to form the initial 2D clusters.

3.3.3 FindClusters

Clustering is performed using two loops; the first takes the highest charge unclustered hit to form a new cluster and the second considers bringing other hits into this cluster. All clustering is performed on the blurred image of the hit map so as to include as many of the particle deposits as possible. If the blurring was correctly implemented this process should successfully cluster together a completely blurred propagating particle. There are many free parameter which can be fixed to alter the clustering.

For a new cluster to be initially formed, the highest charged unclustered hit must have a charge of `MinSeed`; clustering terminates as soon as this is no longer the case. Clusters are grown by looping over all the hits currently forming it and considering other surrounding hits, within a proximity defined by the parameters `ClusterWireDistance` and `ClusterTickDistance`. These are brought into the cluster on condition they meet a minimum charge `ChargeThreshold` and pass a time cut, which ensures that the new hit occurred within a certain time, `TimeThreshold`, to any other previously clustered hit. Note fake hits by definition always pass the time cut.

Once these loops have groups as many hits together into different clusters, a couple of additional steps are performed to ensure the clustering is smooth and as isotropic as possible. First, ‘holes’ (unclustered bins encompassed by a cluster) are filled in; these bins are included subject to the time cut and a cut on the minimum number of neighbours (adjacent clustered hits) `NeighboursThreshold` they must have to be clustered. Finally, ‘peninsulas’, hits which have too few neighbouring hits in the cluster, defined by `MinNeighbours`, are removed to achieve a smooth cluster edge.

Output clusters are only kept if they meet a minimum size requirement (minimum number of hits), `MinSize`, so this check is performed initially at this point. Any clusters not meeting this requirement then they are removed and the bins considered unclustered again. Note `MinSize` refers to the minimum size measured by ‘real’ hits and so this will again be ensured before returning the final output clusters from **ConvertBinsToClusters**.

3.3.4 ConvertBinsToClusters & ConvertBin(s)ToRecobHit(s)

The final stage of the Blurred Clustering algorithm involves converting the clusters of bins from the **FindClusters** method into clusters of `recob::Hits`. During the course of this conversion, fake hits are removed and the final clusters are made solely from initial hits present from the start. This is controlled by the **ConvertBinsToClusters** function.

The clusters are looped over and for each one the method **ConvertBinsToRecobHits** is applied. This takes the bins to be converted and the unblurred hit map as a TH2 histogram and returns vectors of `recob::Hits` to be saved by **ConvertBinsToClusters**. The bin collections are looped over and the function **ConvertBinToRecobHit** used to search the histogram for each bin; if found then the corresponding `recob::Hit` is returned to be placed in the output cluster, if not then this fake hit is removed at this point.

The final output is of the form `std::vector<art::PtrVector<recob::Hit>>` and is returned to the `BlurredClustering_module.cc` producer module. At this point, the clusters are converted to `recob::Cluster`s and placed in the event record for use in proceeding algorithms.

3.4 User-Defined Parameters

The parameters which can be configured to control the clustering have been mentioned previously and are listed and briefly described in the present section. The nominal values which have been chosen for use in DUNE 35-ton shower reconstruction are shown in parentheses.

- `int BlurWire` (6). How far to blur in the wire direction. Note, as discussed in section 3.3.2, this will be scaled dynamically by the relative directionality in the wire dimension.
- `int BlurTick` (12). How far to blur in the tick direction. Note, as discussed in section 3.3.2, this will be scaled dynamically by the relative directionality in the tick dimension.
- `double SigmaWire` (4). The standard deviation of the Gaussian kernel (wire dimension) used in the convolution to produce the blurred hit map. Note, as discussed in section 3.3.2, this will be scaled dynamically by the relative directionality in the wire dimension.
- `double SigmaTick` (6). The standard deviation of the Gaussian kernel (wire dimension) used in the convolution to produce the blurred hit map. Note, as discussed in section 3.3.2, this will be scaled dynamically using relative directionality in the tick dimension and the width of each hit in time.
- `int MaxTickWidthBlur` (10). The maximum factor used to scale the Gaussian in the tick dimension due to the width of each hit in time.
- `int ClusterWireDistance` (2). How far away, in the wire direction, from clustered hits to consider other potential hits for clustering.
- `int ClusterTickDistance` (2). How far away, in the tick direction, from clustered hits to consider other potential hits for clustering.
- `int NeighboursThreshold` (0). The number of neighbours (previously clustered hits) needed for an unclustered bin to be brought into a cluster.
- `int MinNeighbours` (0). Minimum number of neighbours needed to keep a clustered hit in a group after all the clusterings have been performed.
- `int MinSize` (2). Minimum size (number of real hits) needed for a cluster to be saved as a final output cluster.

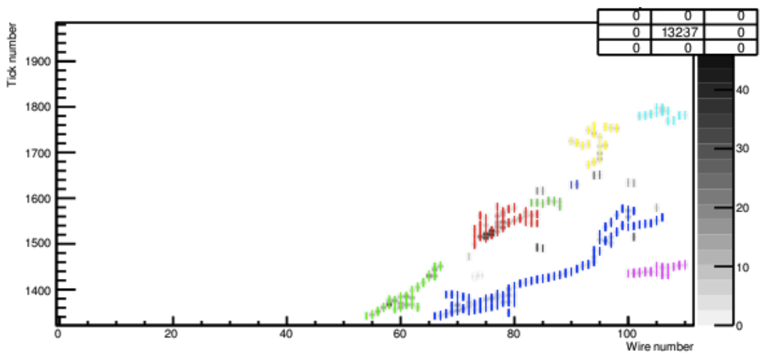


Figure 4.1: Example of a plane where particle reconstruction is fragmented after applying Blurred Clustering.

- `double MinSeed` (0.1). The minimum charge required of the highest charged unclustered hit to initiate a new cluster.
- `double TimeThreshold` (500). Maximum time difference allowed between an unclustered hit and the temporally closest hit in the cluster before bringing the hit into the group.
- `double ChargeThreshold` (0.07). Minimum charge required of a hit before it can be clustered.

Varying these parameters change the effect of the reconstruction significantly so tuning is necessary before applying the clustering to a particular channel. This will be discussed in section 5.

4 Cluster Merging Technique & Track Shower Separation

NOTE: this section is currently outdated. Will be updated soon!

The Blurred Clustering technique, discussed in section 3, is very successful at forming initial 2D clusters within the LArSoft framework. However, due to the extremely fine-grained nature of LAr detectors, these clusters tend to be fragmented and will separate an object into multiple different 2D clusters (see for example the event display shown in Fig. 4.1). This issue is not unique to this particular clustering method but exists in many of the various methods in LArSoft. To address this, a Cluster Merging algorithm has been developed; this is the topic of the present section.

An overview is provided in section 4.1 before the method is discussed in detail in section 4.2. The user-defined parameters are contained in section 4.3.

4.1 Overview

Fragmented clusters associated with the same track are obvious to the human eye because they form a straight line across the clustering space. Combining these together to make complete tracks/showers from the deposits of a particular particle is the aim of the Merging algorithm. It makes use of a Principal Component Analysis (PCA) to determine whether or not two tracks are sufficiently aligned to justify their merger, and also takes into account other factors such as the size of the clusters and their separation.

There are a few parameters which can be used to tune the effect of this algorithm; these will be quoted in `typewriter` font and listed for reference in section 4.3.

4.2 Algorithm Details

The algorithm is designed to run over the output of the Blurred Clustering (and other clustering) functions (`std::vector<art::PtrVector<recob::Hit>>`) and provides output clusters in the same format. As input, it takes a reference to two of these containers – one filled with the input clusters and an empty one in which to place the merged clusters – and the plane, TPC and cryostat being considered.

The algorithm itself consists of two nested loops and is analogous to the FindClusters method in Blurred Clustering (section 3.3.3). The first loop takes the largest cluster not yet considered for merging and forms a new output cluster with it; the second loop takes this cluster and recursively considers all unmerged clusters, merging those which pass certain cuts. Each comparison involves calculating the PCA for all the hits in both the main cluster and the trial cluster and using the returned eigenvalue as a measure of how parallel the two clusters are; they pass this cut if this is at least `MergingThreshold`. The size of the trial cluster must be `MinMergeClusterSize` for it to be considered for merging and the separation cannot be greater than `MaxMergeSeparation`. The separation is calculated for the beginning of the main cluster and the end of the trial cluster, and vice versa, and passes if either distance is within the acceptable range. If all cuts are passed, the cluster is merged into the main cluster and is used to try to merge further clusters. This carries on until all clusters have been considered and each is now clustered again, either by itself or with others.

It should be noted this algorithm is recursive; merged clusters are themselves compared to all other clusters. This could possibly result in a situation where two clusters are not considered parallel enough to merge initially, but if one is merged with a third and the PCA recalculated, it may now be eligible for merging.

Fig. 4.2 shows the same event as in Fig. 4.1, which was used to motivate the Cluster Merging technique, after the MergeClusters algorithm has been applied. The improvement in the reconstruction as a result of this algorithm is very noticeable and justifies its inclusion in the Blurred Clustering reconstruction.

The output clusters are returned to the producer module `BlurredClustering_module.cc`

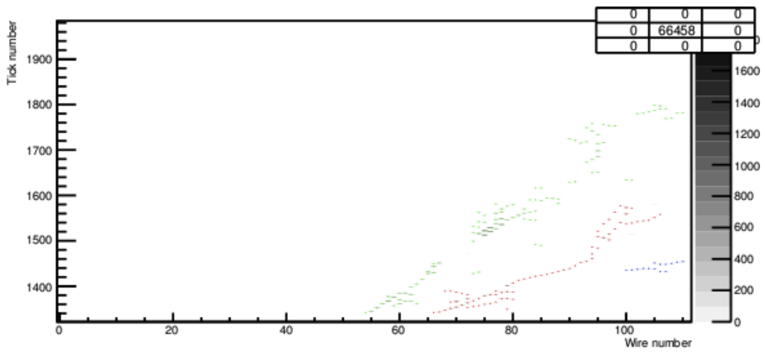


Figure 4.2: The event shown in Fig. 4.1 after the MergeClusters algorithms has been applied. The huge improvement in the reconstruction, due to the merging, can be clearly seen.

where they are converted to `recob::Clusters` and placed in the event record for use by the following modules.

4.3 User-Defined Parameters

The parameters which may be used to control the merging are listed here along with the values chosen for use in DUNE 35-ton shower reconstruction, shown in parentheses.

- `double MergingThreshold` (0.985). The minimum eigenvalue needed from a PCA calculation involving all hits from both clusters before they are merged.
- `int MinMergeClusterSize` (2). The minimum size a cluster needs to be before it can be considered for merging.
- `double MaxMergeSeparation` (30). Maximum separation allowed between clusters before merging can proceed.

5 Configuring

Blurred Clustering provides the user with many free parameters which must be fixed in order to define the type of reconstruction required. This is not a straight forward task but this section contains some ideas and hints which proved useful when tuning the algorithm initially for use in DUNE 35-ton shower reconstruction.

Use a small sample

Generate a small MC sample with typical events needing to be reconstructed and just concentrate on a few (∼10-20) events. Once the events are familiar it will become obvious which parameter values resolved certain parts better than others.

Use the debug pdf

The most useful tool when tuning the individual parameters was the debug pdf, described in section 2.3. This shows the clustering process in stages; the initial hit map, the blurred hit map, the initial clusters and the output clusters. It allows individual tuning to be carried out on the various parts of the algorithm separately without too many parameters interfering at once. First, ensure the blurring looks optimal by tuning the radii and sigma parameters and observing the effect in the debug file. Only then does the clustering need to be considered, initially by adjusting the clustering radii and then by fine tuning with the other clustering parameters.

Tune the important parameters first

The most important parameters are those affecting the blurring and the clustering distances. Most time should be spent with these to find the most optimum combinations; parameters such as `MinNeighbours` are only useful in specific circumstances. Variables such as the `ChargeThreshold` and `MinSeed` are unlikely to change greatly from their default values but can be adjusted if necessary. `MinSize` is really only useful for how the output clusters should look and will not affect the reconstruction itself at all.

Consider merging last

The merging stage should be the final thing to be considered, and only once the clustering appears satisfactory. Use the event viewers to determine if the algorithm is required and turn it on if necessary. The parameters require very fine tuning and there will need to be compromises between different types of events for which its use is a huge improvement and others which suffer slightly. The `MergingThreshold` parameter is the most important and should be considered first; the other parameters have a much lesser effect. `MinMergeClusterSize` will rarely affect the merging and is useful only if output cluster size and computational expense are factors, whilst `MaxMergeSeparation` will be the final thing to set and only after using the event viewer to decide that such a threshold is necessary in this particular instance.

There will be other considerations when tuning the algorithms but these guidelines should allow the process to not require huge amounts of time and avoid circular thought processes. If further parameters prove useful then the author would be interested to know what they are!

6 Performance

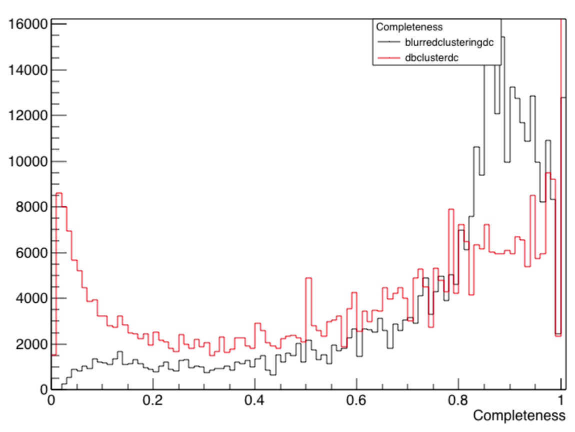
The effectiveness of a particular clustering method can be quantified with the following metrics:

- Cleanliness: How clean a cluster is, i.e. the ratio of signal hits (hits from the particle with which the cluster is associated) in a cluster to the total number of hits.
- Completeness: How complete a cluster is, i.e. the ratio of the number of hits from the particle with which the cluster is associated which are clustered to the total number of hits the particle left.

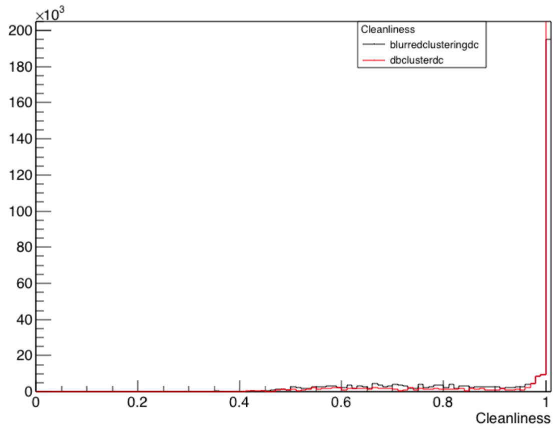
- Efficiency: How well the clustering deals with specific event topologies. This requires a cut; the algorithm is tuned on a π^0 sample and so here this stipulates at least two clusters in a particular view, each at least 50% complete. The efficiency for a particular variable is the ratio of its distribution for events which pass the cut to those which do not.

The distributions of these variables, weighted by cluster size, are shown below for the reconstruction of 500 events of a cosmic π^0 sample. For comparison, the same distributions for the `dbcluster` algorithm (also within LArSoft) are also shown.

Fig. 6.1 shows the completeness and the cleanliness of the clusters found by the two reconstruction techniques. It can be seen in general Blurred Clustering performs much better; it has a higher overall completeness and fewer clusters with a low completeness. The cleanliness is in general very high for both methods, confirming the understood behaviour that each preferentially splits up tracks/showers into smaller but very clean clusters.



(a) Completeness



(b) Cleanliness

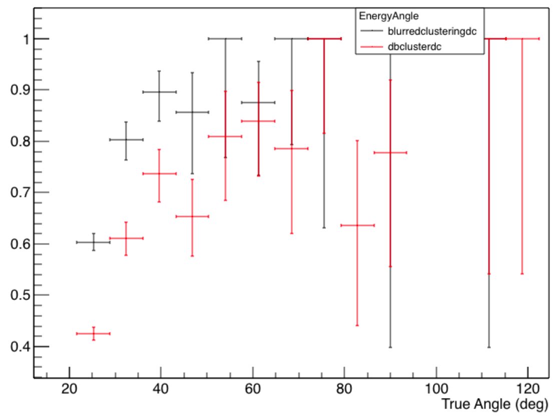
Figure 6.1: Completeness and Cleanliness of the clusters found after running the `blurredclustering` and `dbcluster` reconstruction over a typcial cosmic π^0 sample

Fig. 6.2 shows the efficiency of the reconstruction for two variables; the decay angle of the π^0 (i.e. the angle between the two decay photons) and the distance from the vertex of the conversion point of each photon. Again, it can be seen Blurred Clustering in general yields superior reconstruction performance, showing a greater efficiency in most bins.

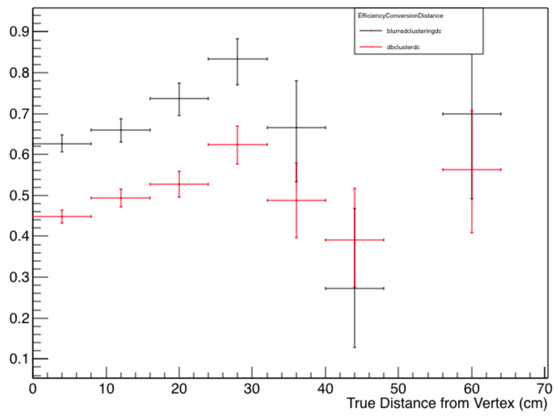
These histograms were made using the art analyser module larreco/ClusterFinder/ClusteringValidation_module.cc which is currently set up to analyse a π^0 sample.

7 Summary

Blurred Clustering has been shown to provide excellent reconstruction for a π^0 sample in DUNE 35-ton. It is a flexible algorithm and can be tuned to work over many different event topologies.



(a) Angle



(b) Photon conversion distance

Figure 6.2: Efficiency of the variables decay photon angle and photon conversion distance from vertex for the reconstruction of a cosmic π^0 sample for both `blurredclustering` and `dbcluster`.

This document provides an introduction to the algorithm and a detailed description of its main methods for use within LArSoft. Development and improvements are ongoing and any changes will be reflected in this note. Hopefully the clustering will find use in plenty of other different analyses and experiments!

References

[1] https://en.wikipedia.org/wiki/Gaussian_blur

[2] http://hdl.handle.net/1721.1/76980