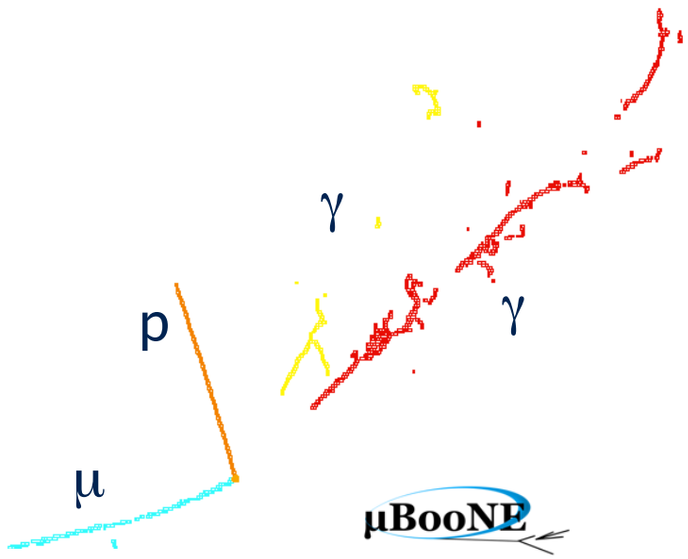


larpandoracontent v03_07_00

J. Anthony for the Pandora Team

LArSoft Coordination Meeting

June 6, 2017

Outline

This talk outlines the changes proposed for larpandoracontent v03_07_00:

- Eigen dependency added and dealt with in build tools
- 2D/3D sliding linear fits now use Eigen to determine coordinate system
- Shower direction convention updated so that principal axis points away from the vertex
- PcaShowerParticleBuilding algorithm moved from larpandora to larpandoracontent
- LArCaloHit object added to hold track, shower, other and Michel classification probabilities
- Added an algorithm for an SVM-based approach to vertex selection using the SVM interface added in v03_05_00
- Added an algorithm for SVM-based track/shower ID
- SVM data files located from FW_SEARCH_PATH using cet::search_path functionality

The changes are in feature branches named 'feature/larpandoracontent_v03_07_00' in larpandoracontent, larpandora, uboonecode, uboonedata and dunetpc

Eigen dependency and usage I

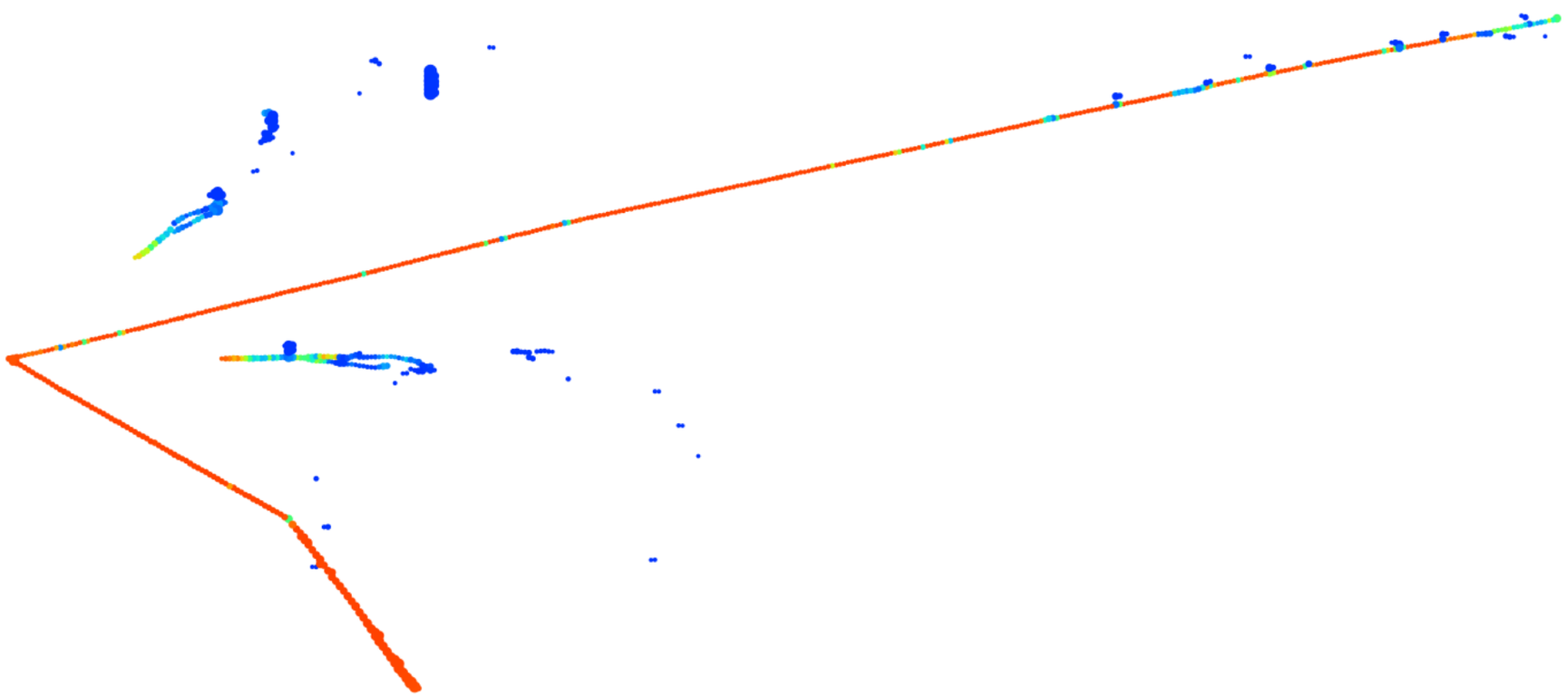
- Eigen is a header-only C++ library for linear algebra, facilitating e.g. PCA
- 2D and 3D sliding linear fit objects in Pandora need to determine a coordinate system against which to describe the fit
- Extremal points of the Cluster were previously used to determine this axis, which usually form a good basis
- A bad choice of axis can lead to a poor fit:
  - The Cluster position could be effectively multivalued on a bad projection
  - Traversing small distances across the fit can project to large changes along the Cluster, so accuracy is lost
- To improve quality of the coordinate system, Eigen can be used to determine the principal axis instead
- **An Eigen dependency has been added to larpandoracontent**
- Changes to build mechanics are working nicely – fairly minimal since Eigen is header-only

Eigen dependancy and usage II

- PcaShowerParticleBuilding algorithm already depended on Eigen so had been housed in larpandora rather than larpandoracontent to avoid this dependency
- The dependency has been adopted and this algorithm has now been moved to larpandoracontent
- The chosen shower axis direction was following Eigen's default axis direction convention (low x to high x if not single-valued in x; else low y to high y etc)
- Analysers didn't like this convention – so this has been altered such that **the direction of the axis always points away from the shower vertex**
- We assume that the reco centroid is farther from the start of the shower than the vertex

Hit-based track/shower ID

- A new object, lar_content::LArCaloHit, has been added that endows pandora::CaloHit objects with normalised track-like, shower-like, or none/other probabilities – plus a Michel-like probability
- The probabilities are set during instantiation of the LArCaloHit object and can then be used for pattern recognition. By down casting CaloHits to LArCaloHits, we can access these additional properties
- This change facilitates the ongoing development of Hit-level track/shower/other(/Michel) ID, in collaboration with Robert and Dorota



Pandora SVM framework

- SVM framework was added In larpandoracontent v03_05_00 to generalise the treatment of SVMs in the now-complete SVM vertex selection algorithm
- This comprises two parts:
  - **LArSupportVectorMachine object:** provides the fundamental mechanics
  - **LArSvmHelper class:** automates some low-level work to facilitate high-level usage
- The framework allows already-trained SVMs to be used inside Pandora to make binary classification decisions
- Training of the SVMs must be done using an external package – but mechanics for producing training sets in the right format is provided
- The trained SVMs take the form of an XML document with a set of parameters, such as the type of kernel to use and the number of features, followed by an arbitrarily long list of support vectors with the same dimensionality as the feature space

LArSupportVectorMachine I

- The LArSupportVectorMachine object provides the mechanics for initialising the SVM from the XML data file, extracting properties of the initialised SVM, asking for/defining the kernel function and performing classification
- Contents of the data file and the state of the SVM are checked for consistency and an exception is raised if there is an issue:
  - Number of features
  - Size of support vectors
  - Whether the SVM has been initialised before usage
  - Sizes of the mu/sigma vectors, used to standardise the data if necessary
- The object is designed with performance in mind, so C++11 move semantics have been taken advantage of and further analysis of the performance is going to be performed in the near future
- Common kernels (linear, quadratic, cubic, Gaussian RBF) are provided and can be set via an enum, or a custom kernel can be defined by passing a function pointer or using a lambda function

LArSupportVectorMachine II

- Declared alongside the SVM object is an abstract class template called SVMFeatureTool, which inherits from AlgorithmTool
- SVMFeatureTool is an AlgorithmTool with a virtual 'Run' method:
  - First argument is a vector of doubles (the vector of features to be appended by the tool)
  - Remaining arguments are a variadic list of arguments to be passed to the method, templated at the class-level
- The idea is that each algorithm has a set of features, calculated by a set of SVMFeatureTools, all templated on the same input variables – then we can recast the tools and exploit the polymorphism in useful ways
- SvmFeatureTools can be configured from at the setting file level in the same way as AlgorithmTools, allowing different features and different numbers of features to be used without recompilation
- Each feature tool can calculate an arbitrary number of features and append them to the vector – feature dimensionality checking is then performed by the SVM

LArSvmHelper

- The LArSvmHelper provides a number of static methods for facilitating the high-level use of the LArSupportVectorMachine object
- This includes wrapper methods around the classification functionality, allowing variadic lists of features to be automatically concatenated before being passed to the SVM – this is useful since some features may be calculated via feature tools, some by feature tools of different templating, and simpler ones by the algorithm itself
- Functionality is provided for initializing the list of feature tools from the AlgorithmTool pointers in the algorithms' ReadSettings methods
- The ProduceTrainingExample method appends lists of features to a designated file in a standard format for training an SVM
- Finally, there are methods for calculating and returning lists of features, given the vector of upcast SvmFeatureTools and the input arguments, which are common to the feature tools by design
- The helper class relies fairly heavily on method templating and implements compile-time type-checking via static_asserts where possible

SVM data files

- Each SVM decision made in Pandora requires a data file that parametrises the SVM
- These files are typically ~a few MB in size
    - e.g. SvmVertexSelection algorithm's data file is 7.0MB and comprises two SVMs
- The data files are specific not only to the problem but also to the input data, so we require separate data files for e.g. MCC7/MCC8, as well as different files for MicroBooNE/DUNE
- As per email discussion, the MicroBooNE SVM data files will live in uboonedata product, (uboonedata/PandoraData) and similarly in dune_pardata for DUNE
- Within the SVM-based algorithms, the data files are then located using the cet::search_path functionality

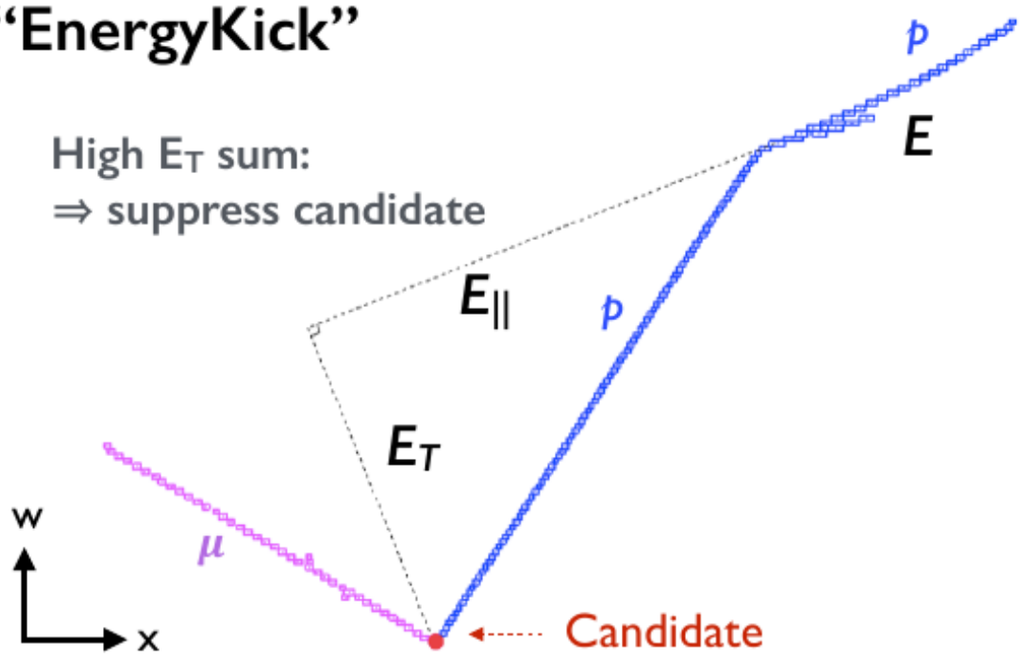
Vertexing in Pandora

- Vertexing in Pandora occurs at the 2D stage of the reconstruction and consists of 2 parts:
  - **Candidate creation:** features of the 2D clusters are used to create a large number of 3D candidate vertices
  - **Vertex selection:** the vertices are scored, based on a number of variables, and the highest-scoring one is chosen
- The previously-available vertex selection algorithms were the EnergyKick and HitAngle algorithms
- To combine multiple scores, e.g. beam deweighting, energy kick and asymmetry in the EnergyKick algorithm, we create a probability-like quantity for each and take their product to make a final 'probability', with some parameters to be manually tuned

SVM-based vertexing motivation I

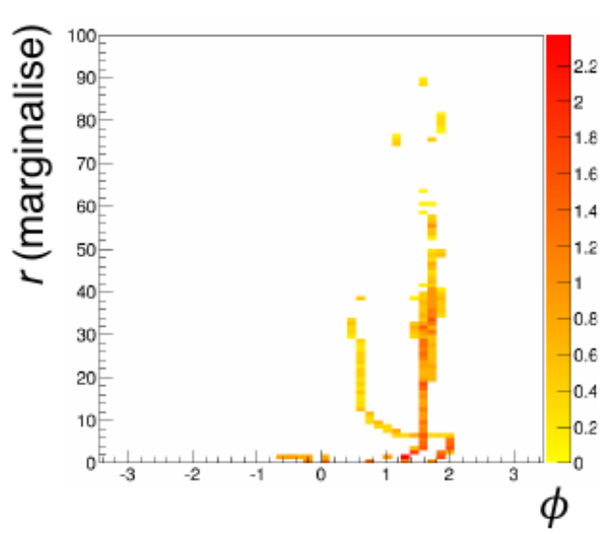
- HA uses the r/phi distribution of the Hits in the vicinity of a candidate to make a score
- EK uses the transverse component of each cluster energy wrt the vertex candidate to work out the size of the energy 'kick' required – plus an asymmetry score to suppress candidates that split clear tracks
- Both algs use the 'beam deweighting score' to favour upstream candidates

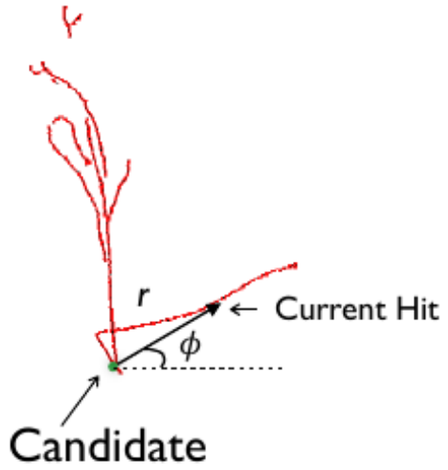
"EnergyKick"



~~• Now the default algorithm~~

"HitAngle"

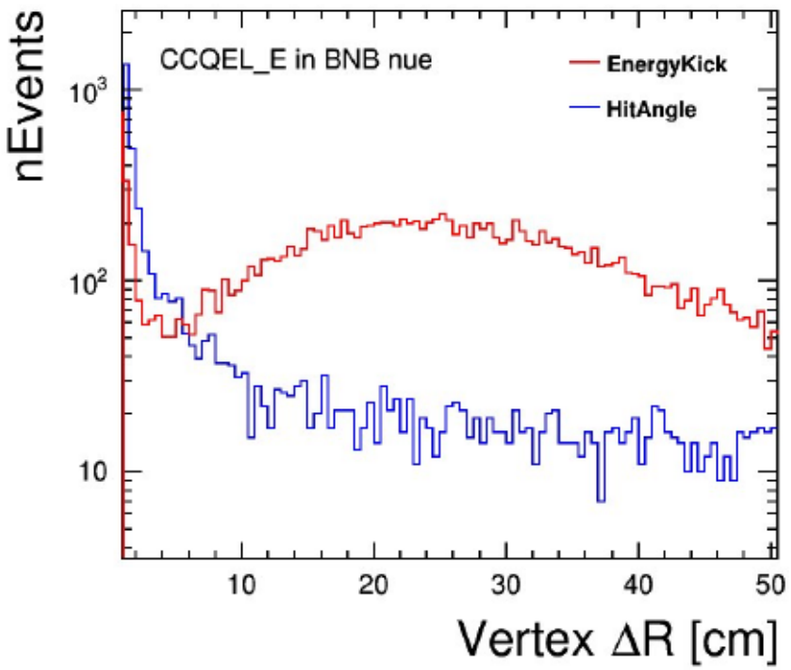


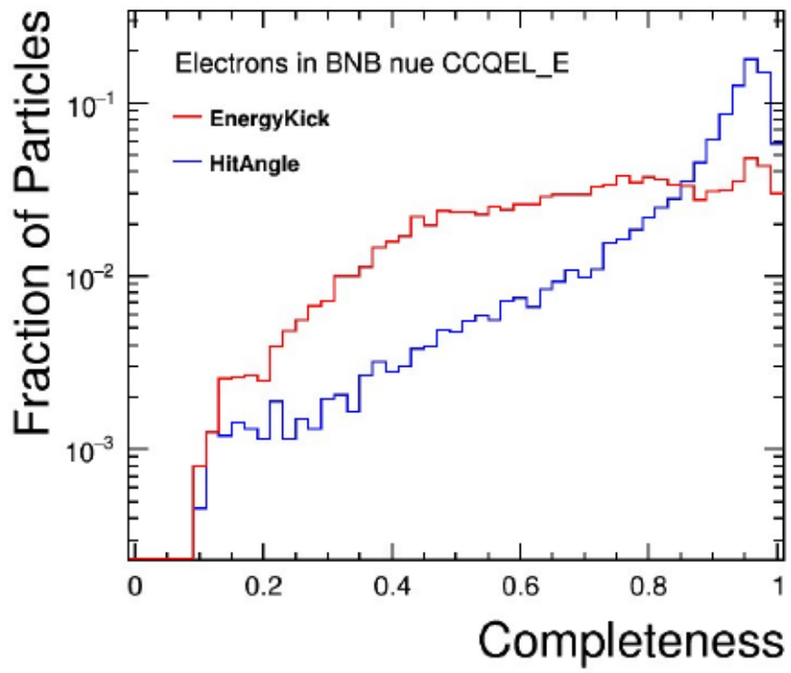


- As used for Neutrino 2016

From DocDB 6753-v1

SVM-based vertexing motivation II






```
                                                    nReco particles matched to electron in CCQEL_E
EnergyKick algorithm:
-CCQEL_E,     nEvents 29,421,  fCorrect 48.7%   ------→  |0: 0.9%|, |1: 48.7%|, |2: 41.0%|, |3+: 9.4%|
-CCQEL_E_P,   nEvents 35,243,  fCorrect 65.3%                          - Particle splitting! -
-CCQEL_E_P_P, nEvents  1,797,  fCorrect 70.6%

HitAngle algorithm:
-CCQEL_E,     nEvents 29,421,  fCorrect 88.8%   ------→  |0: 1.2%|, |1: 88.8%|, |2: 9.1%|, |3+: 0.9%|
-CCQEL_E_P,   nEvents 35,243,  fCorrect 73.3%
-CCQEL_E_P_P, nEvents  1,797,  fCorrect 60.0%

MCCheating algorithm:
-CCQEL_E,     nEvents 29,421,  fCorrect 91.9%   ------→  |0: 1.4%|, |1: 91.9%|, |2: 6.3%|, |3+: 0.4%|
-CCQEL_E_P,   nEvents 35,243,  fCorrect 85.8%
-CCQEL_E_P_P,  nEvents 1,797,  fCorrect 81.6%
```

From DocDB 6753-v1

(MCCheating = cheated vertex, not just cheated selection > achievable ceiling here)

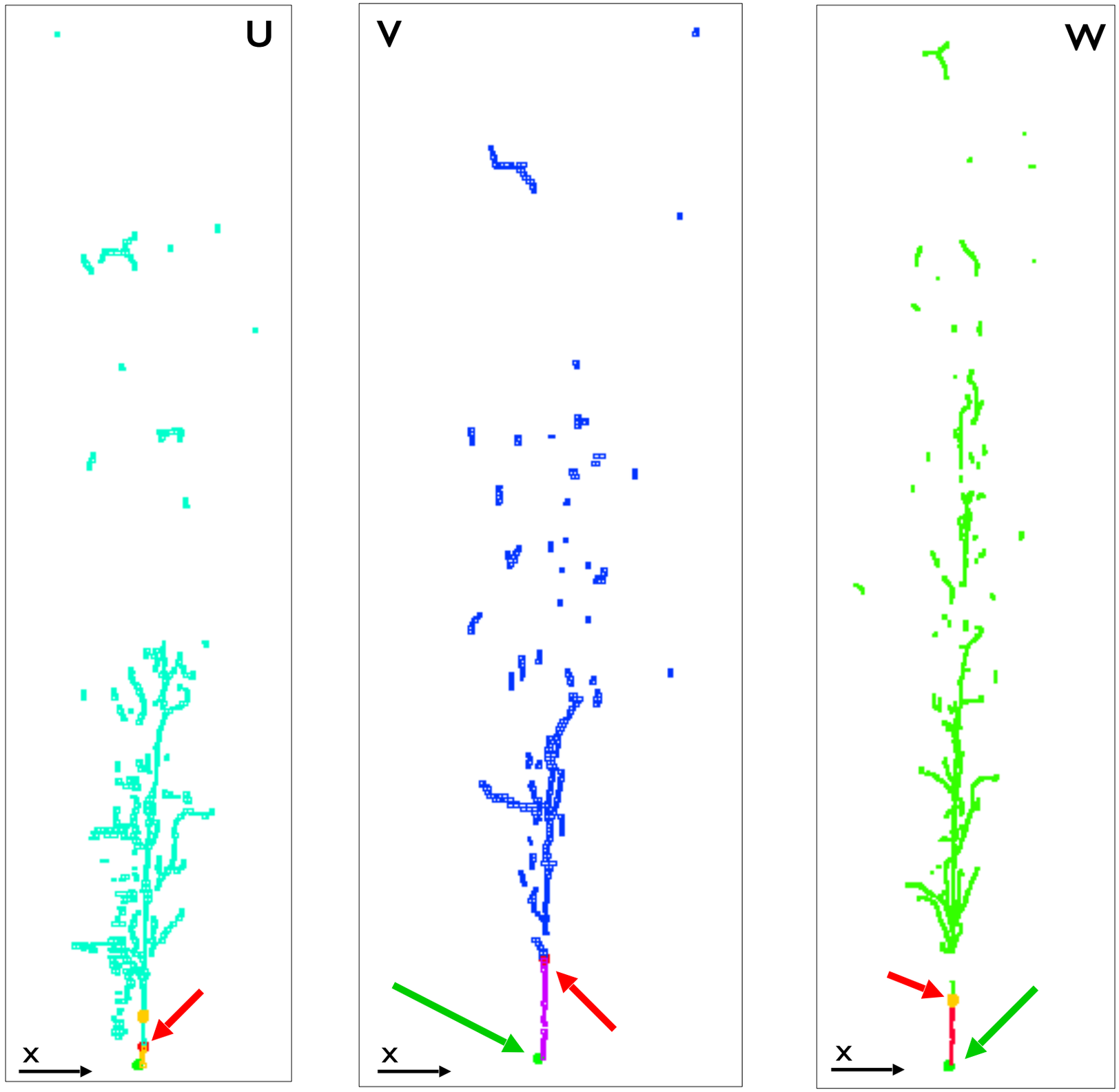
Common pathology

ν_μ ESCATTER ν+e

EnergyKick alg

E_nu = 1363MeV

Incorrect



True vtx, reco vtx

SVM approach

- Meaningfully combining the scores becomes difficult as the score components become more codependent and when different topologies prefer different tunes
- These score components abstract an apt feature space but turning a position in the feature space into a scalar score is difficult – ML can help us here
- SVMs are trained binary classifiers (see backup slides for further description)
- To phrase the problem as binary classification, we train the SVM (details later) to decide which of two vertices is better (using MC information), given a set of features.
- We split the problem into vertex-region-finding and vertex-finding, since the problems are quite different – yields a performance benefit

Feature selection

- Features are based on those used for vertex selection in previous algorithms, plus some event-based features that allow for different topologies to be treated differently
- Simple distance-based shower-like 2D Cluster clustering is used to provide info about candidates splitting showers
- **Region-finding SVM:**
  - **Vertex-based features**: energy kick, beam deweighting, global asymmetry, local asymmetry, shower asymmetry
  - **Event-based features**: event showeryness (proportion of showery-cluster-associated Hits), total energy, volume spanned, longitudinality, number of Hits, number of Clusters, number of vertex candidates
- **Vertex-finding SVM:**

  [same as above plus the vertex-based r/phi feature]

Training set production

Region SVM

- Use a method similar to the EK alg to score all the vertices
- Run down the list of vertices and pick every vertex that is >10cm separated from every other chosen vertex, to define disjoint spherical regions of radius 10cm
- Use the MC information to get the best region (and skip this event if the best region isn't within 10cm of the true vertex)
- Produce the feature sets for all the regions
- For each incorrect region, add a training example comprising the feature set of the correct region and the feature set of the incorrect region, with a 50% probability of each ordering – then class 'true' means the better region was in the first position

Vertex SVM

- Same as the above, except now looking at all the vertices in the correct region and comparing them with the best vertex

We train the SVMs using the sklearn Python package with a Gaussian radial basis function kernel, then package the support vectors and other parameters as an XML file to be read by Pandora

SVM-based track/shower ID

- A new algorithm has been added for performing track/shower ID for use by analyzers
- The algorithm uses the same SVM mechanics to perform binary classification on PFOs, based on a number of features
- The mechanics of using and manipulating the SVM object, as well as producing training sets, for this problem is the same as for the vertexing – all abstracted by the SVM object and associated helper class
- The main difference between these problems is in the judicious choice of features
- Training sets currently produced for MicroBooNE – to be studied for use in DUNE in the future

Track/Shower ID features

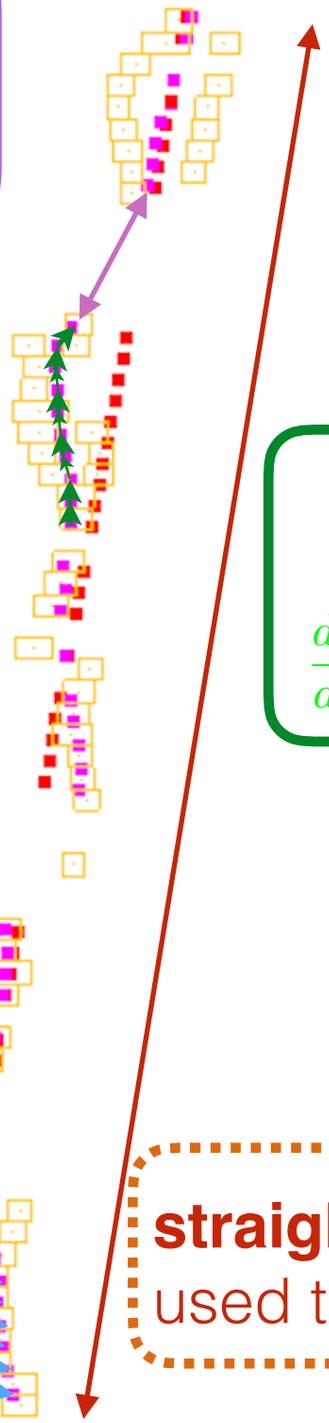
max separation between fit layers

RMS of the sliding linear fit
sum in all layers

difference with straight line
mean and σ of

$$dT = |T_s - T_L|$$

short/Large fit window



True shower hits

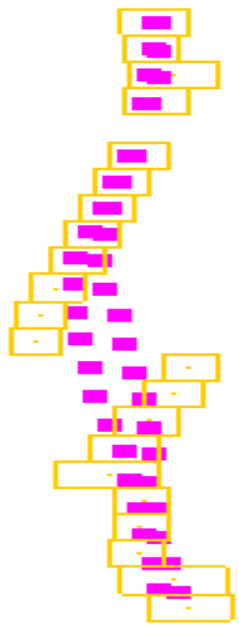
dTdL width
width of fit dT/dL

$$\frac{dT}{dL}_{width} = \frac{dT}{dL}_{max} - \frac{dT}{dL}_{min}$$

straight line length
used to normalise

shower fit width
sum (layer by layer) difference between fits to positive and negative edges

$$\sum dT = \sum |T_P - T_N|$$



distance to vertex

Feature branches

- The changes are in feature branches named '**feature/larpandoracontent_v03_07_00**' in larpandoracontent, larpandora, uboonecode, uboonedata and dunetpc
- Branches are complete and have been tested

Backup slides

SVM references

- Some very detailed notes:

  http://cs229.stanford.edu/notes/cs229-notes3.pdf

- Some easier slides:

  http://www.robots.ox.ac.uk/~az/lectures/ml/lect2.pdf

- scikit-learn SVM description and examples:

  http://scikit-learn.org/stable/modules/svm.html

- scikit-learn SVM documentation:

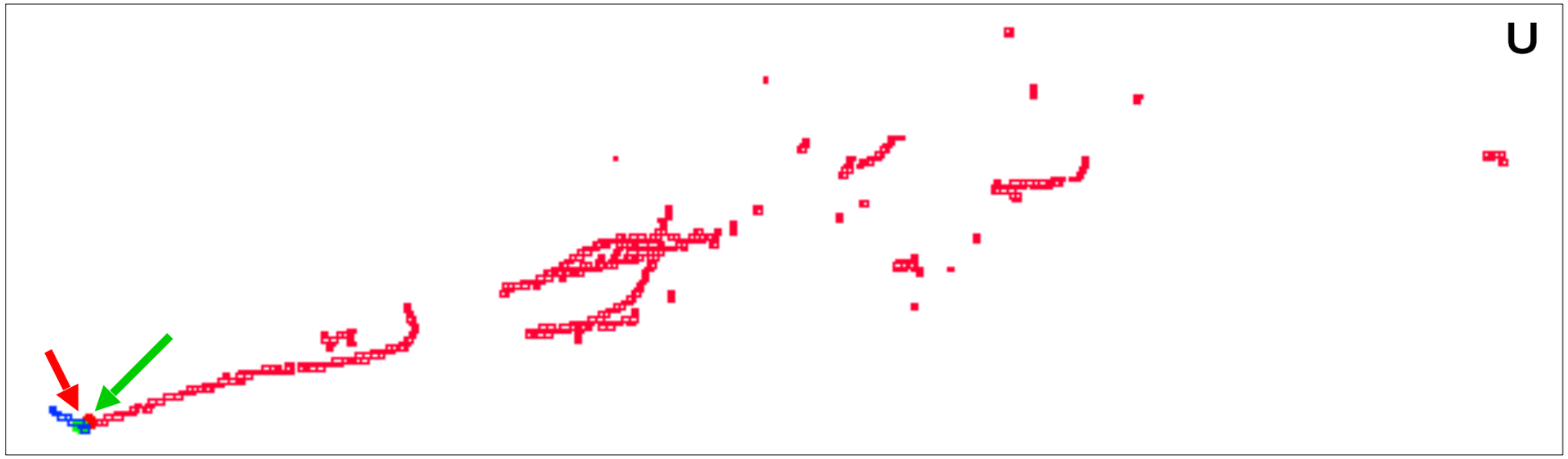
  http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

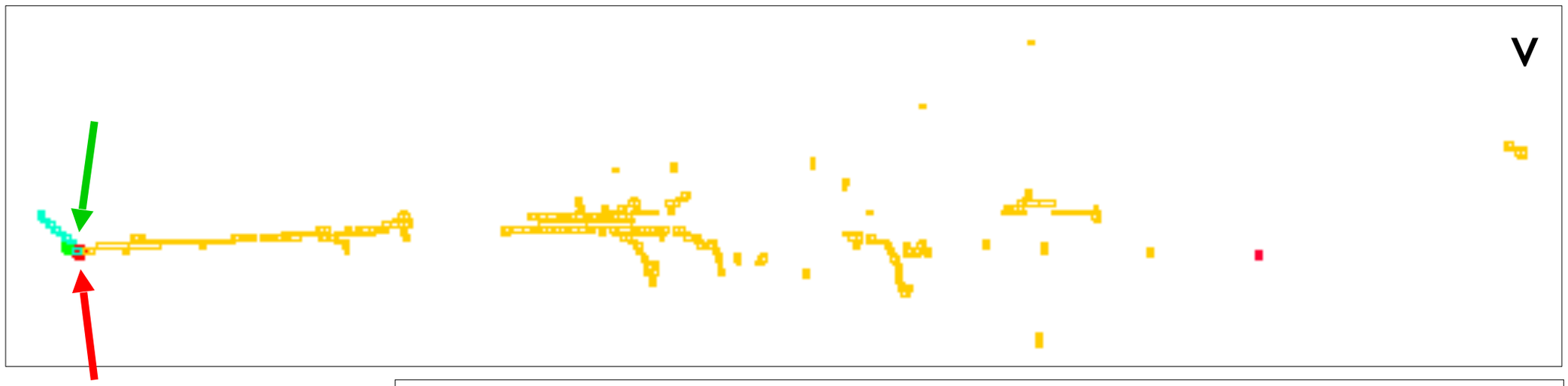
ν_e CCQE e+p

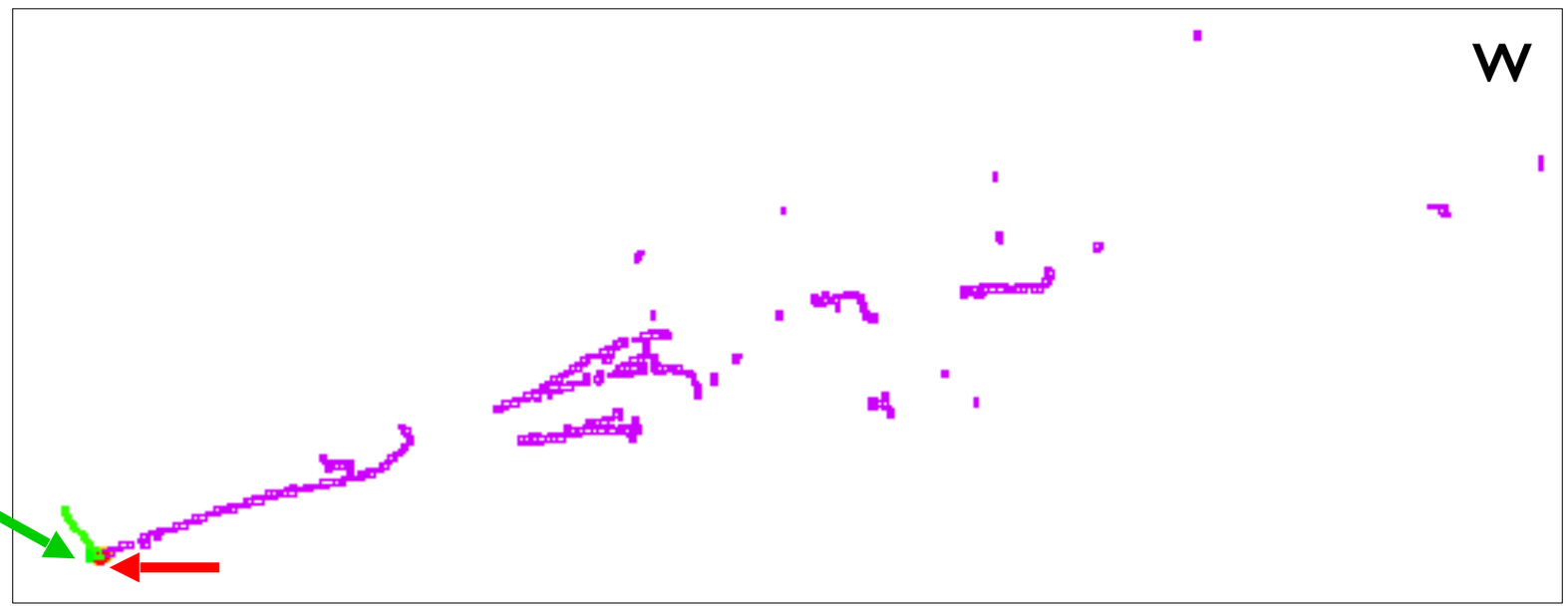
EnergyKick alg

E_nu = 471MeV

Correct







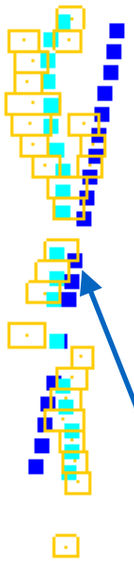
True vtx, reco vtx

Sliding fits graphically

SlidingLinearFits

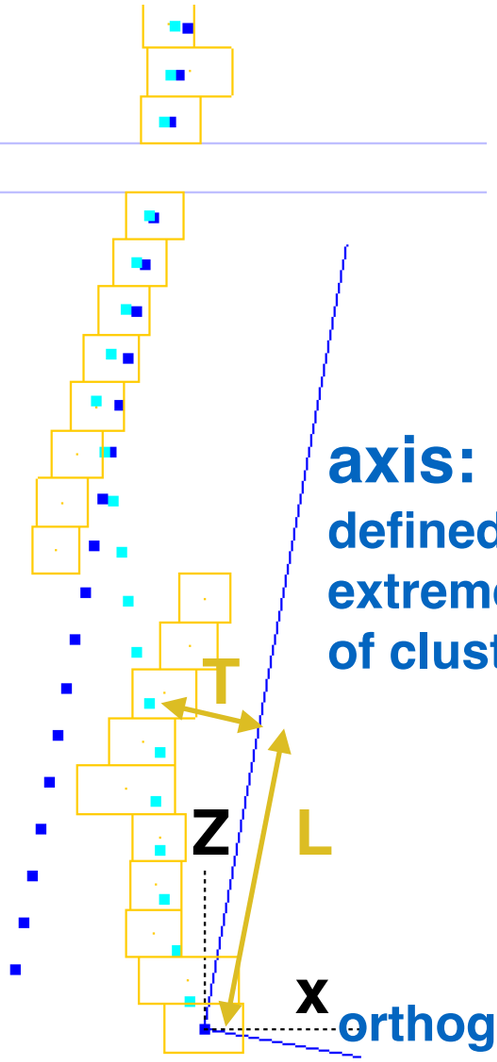
Perform a linear fit taking into account the positions of hits in a (sliding) fit window

Sliding linear fit with a small fit window, following the shape of the cluster



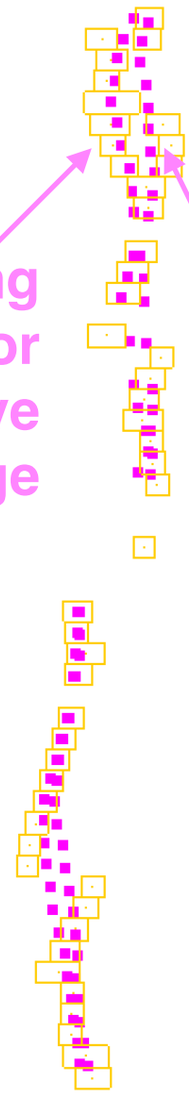
Straight line: using a large fit window, i.e. including all hits in the fit

Local coordinates: L and T



SlidingShowerFit

Perform two sliding linear fits, on positive and negative cluster edges



Sliding linear fit for the negative edge

Sliding linear fit for the positive edge

Pandora LAr TPC Reconstruction

Pandora is an open project and new contributors would be extremely welcome.
We'd love to hear from you and we will always try to answer your questions!

	Contact details:
Framework development	**John Marshall (marshall@hep.phy.cam.ac.uk)** **Mark Thomson (thomson@hep.phy.cam.ac.uk)**
LAr TPC algorithm development	**John Marshall** **Andy Blake (a.blake@lancaster.ac.uk)**
Performance metrics and validation	**John Marshall** **Andy Blake** **Lorena Escudero (escudero@hep.phy.cam.ac.uk)** **Joris Jan de Vries (jjd49@hep.phy.cam.ac.uk)** **Jack Anthony (anthony@hep.phy.cam.ac.uk)**

Please visit https://github.com/PandoraPFA

UNIVERSITY OF CAMBRIDGE

Lancaster University