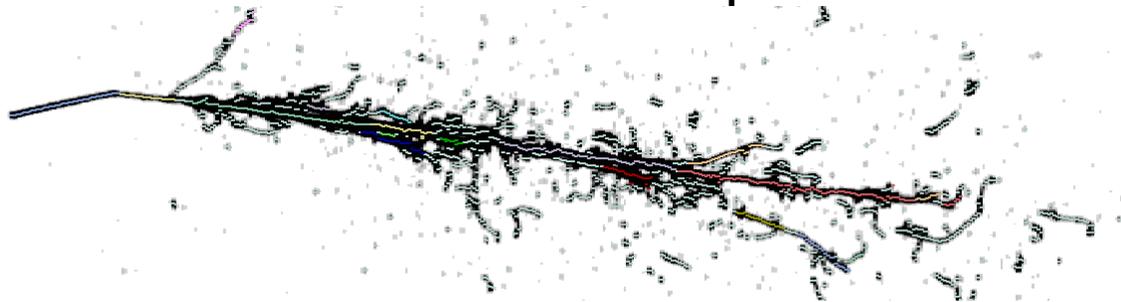


# Speeding up the Hough Transform

Ben Carls

# Status of Hough transform

- Effective at finding lines, yet very slow
- Fortunately, there are many avenues for potential improvements
- The biggest slowdown in the original HoughLineAlg resulted from refilling the accumulator from scratch after every line was found
- First optimization came from subtracting points from the original accumulator after each point was found



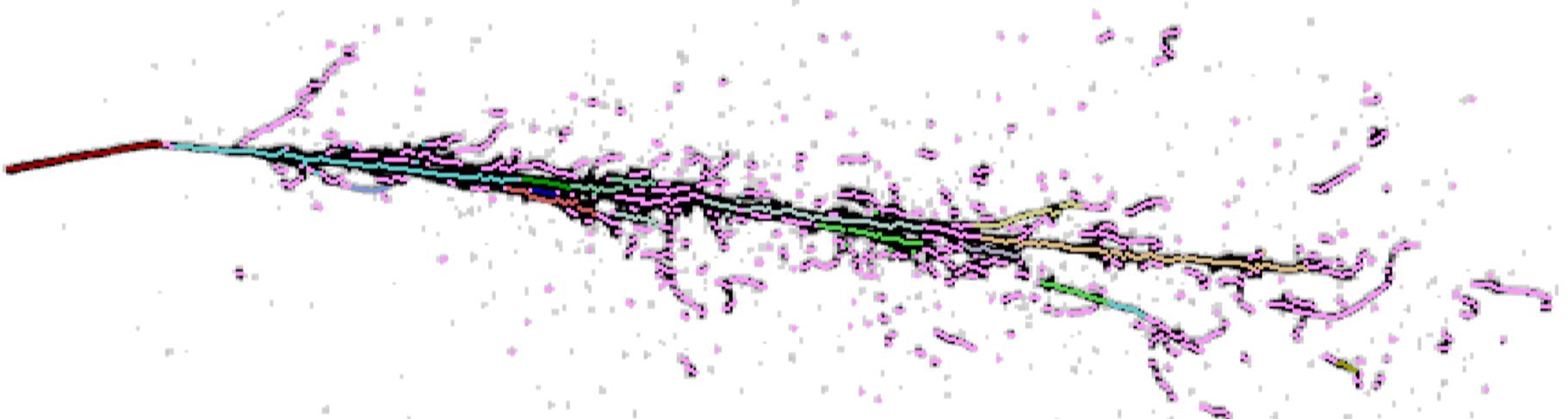
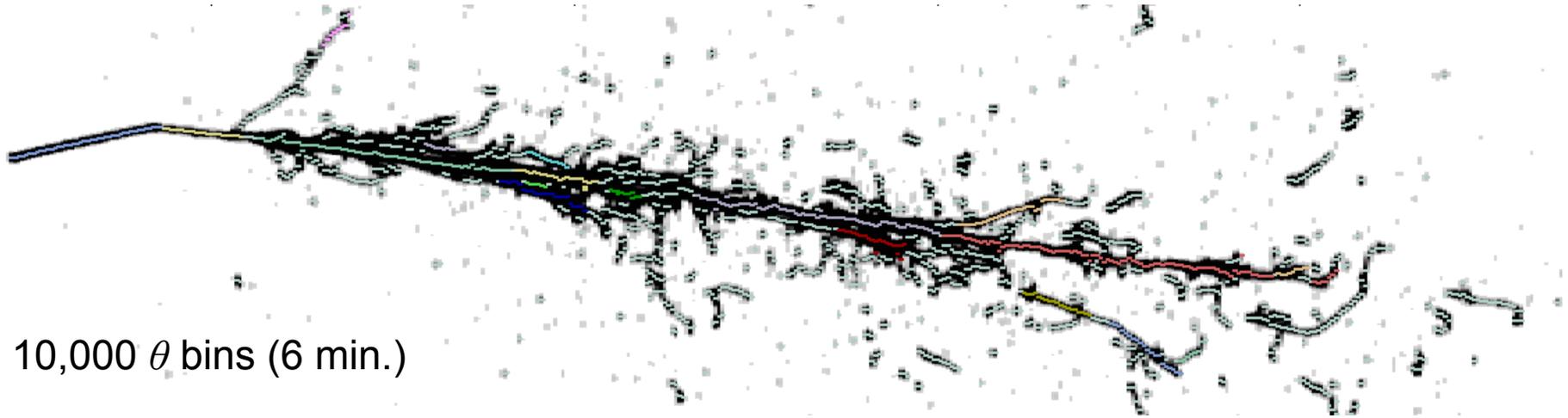
For this event, took processing time from 37 minutes to 6 minutes, even running at double the  $\theta$  resolution

# Going even faster with a PPHT

- Early work using a Progressive Probabilistic Hough Transform (PPHT) has further increased speed
- The algorithm is fairly simple:
  1. Randomly select one hit, remove it from the image
  2. Check if the highest peak in the accumulator modified by the pixel is higher than a threshold cut; if not, go to step 1
  3. Add points along the line, removing them from the image
  4. Remove hits from the accumulator from the line that was found
  5. Repeat step 1 if the image is not yet empty

J. Matas et al., Robust Detection of Lines Using the Progressive Probabilistic Hough Transform, Computer Vision and Image Understanding, Volume 78, Issue 1, April 2000, Pages 119–137

# PPHT performance



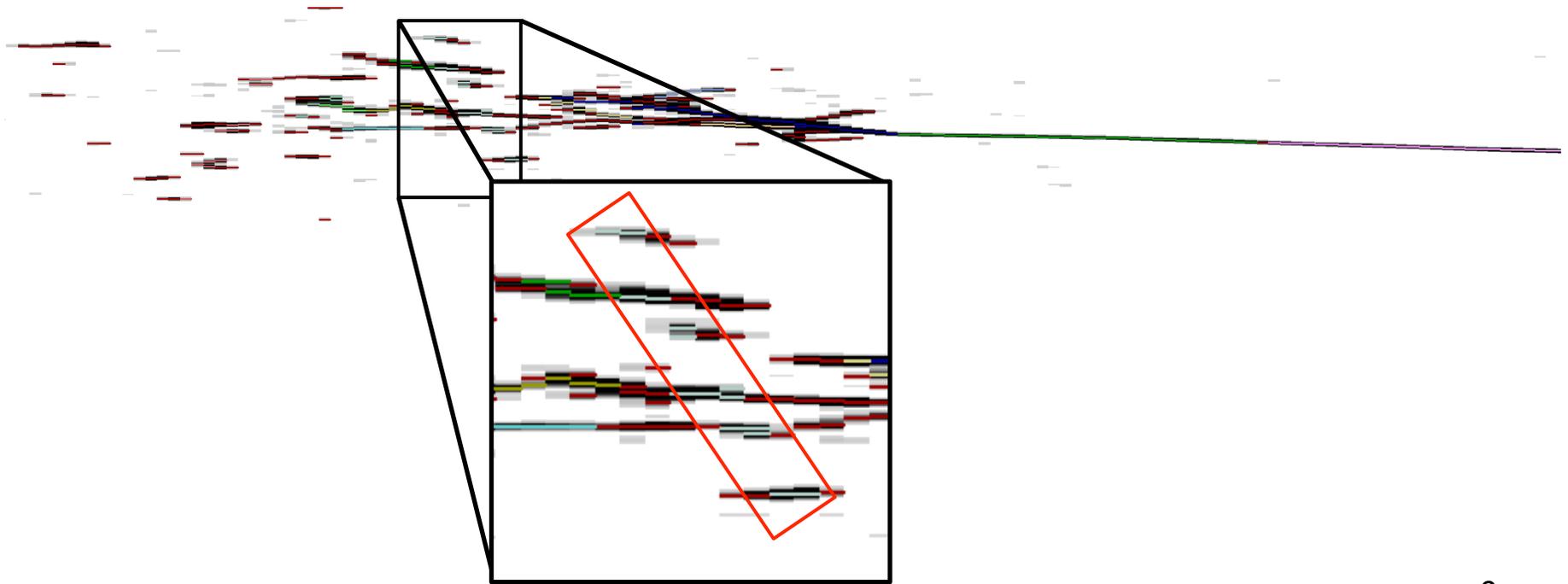
20,000  $\theta$  bins (1 min.)

# Aim of the improvements

- Want to first apply a clustering algorithm (fuzzy clustering) for a rough first pass
- Then apply the PPHT to construct showers and tracks
- The PPHT is only implemented in the HoughClusAlg class, usable by fuzzy clustering code only, could be implemented in HoughLineFinder

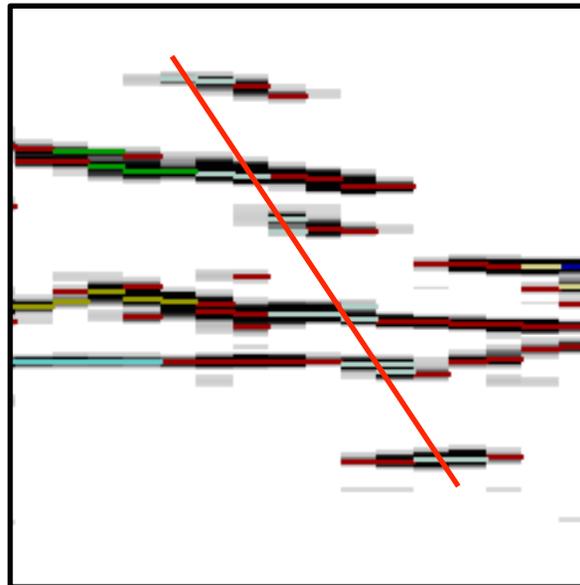
# Dealing with fake lines

- The Hough line finders tend to create lines out of steep, uncorrelated hits



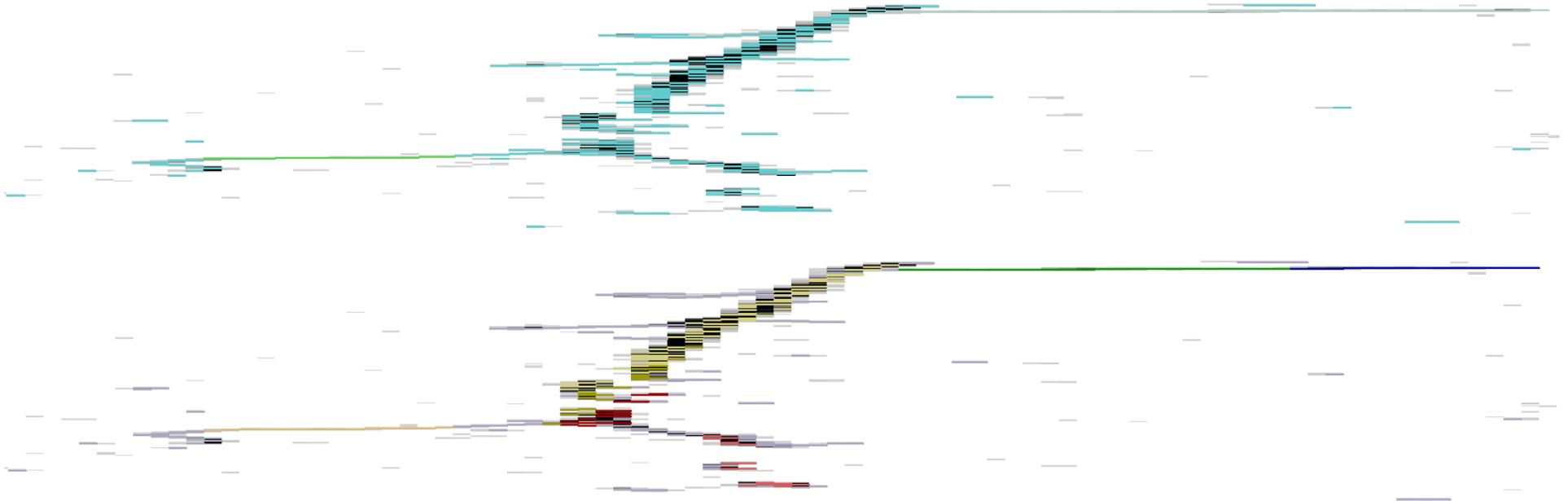
# Dealing with fake lines

- Strategy walks along the line and searches for missing hits
- If too many missing hits are found, the line is vetoed



# Dealing with fake lines

- Still needs some slight tuning, occasionally rejects good lines
- Most of the lines in this shower were rejected



# Other modifications

- A few of the functions in the HoughClusAlg and the HoughLineAlg classes were shared
- Created a base class HoughBaseAlg for HoughClusAlg and HoughLineAlg to inherit from
- Everything is in the repository

# Hough transform

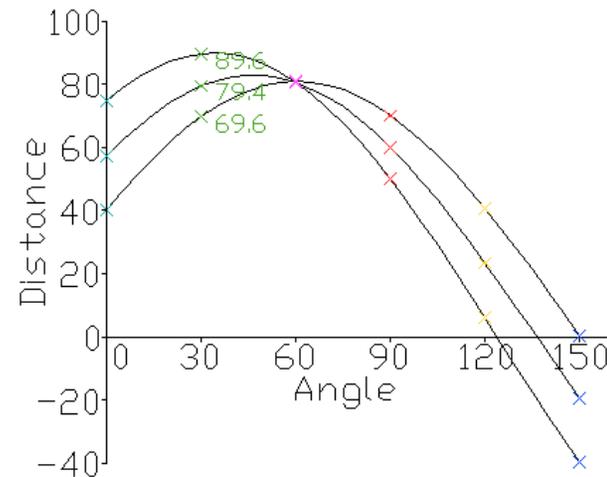
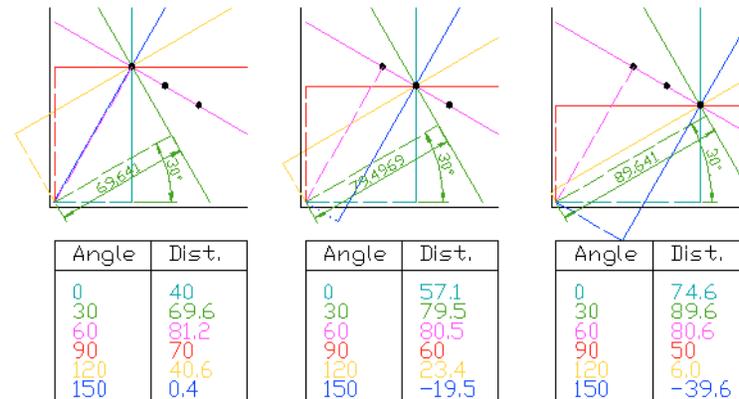
Use the parameterization:

$$y = \left( -\frac{\cos \theta}{\sin \theta} \right) x + \left( \frac{r}{\sin \theta} \right)$$

Fill a matrix (accumulator) in  $(r, \theta)$  space with:

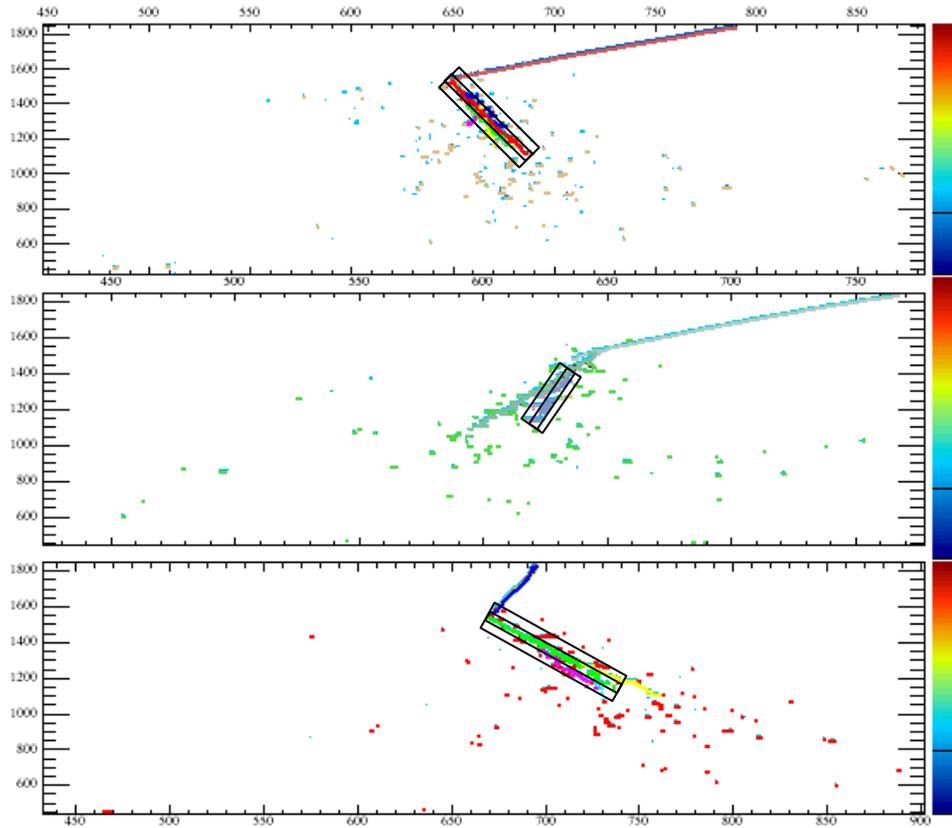
$$r(\theta) = x_0 \cdot \cos \theta + y_0 \cdot \sin \theta$$

We then search the accumulator for the most populated bin, rather computationally demanding



The accumulator, binned

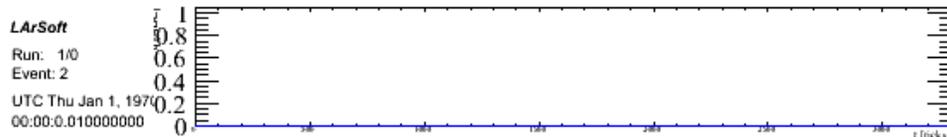
# The older method of dealing with Hough lines in electron showers



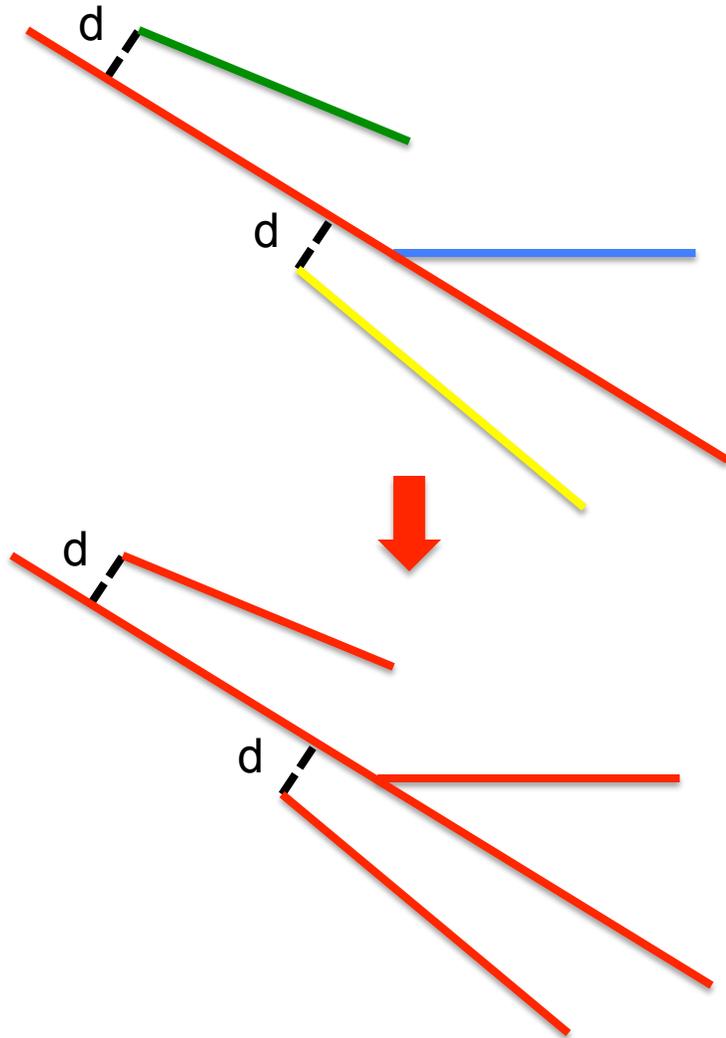
Hough line finder identifies lines in electron showers, need to veto

Examining area around lines and checking isolation, applying a cut

This cut is applied after the lines have been merged



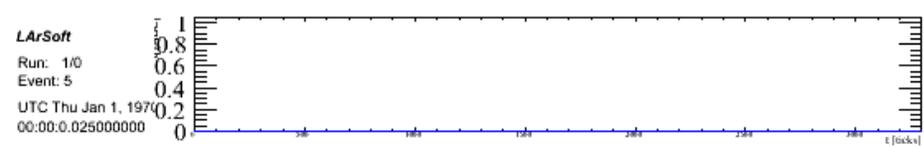
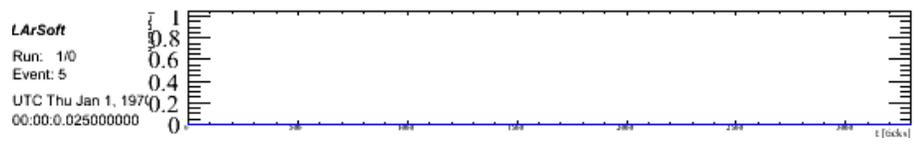
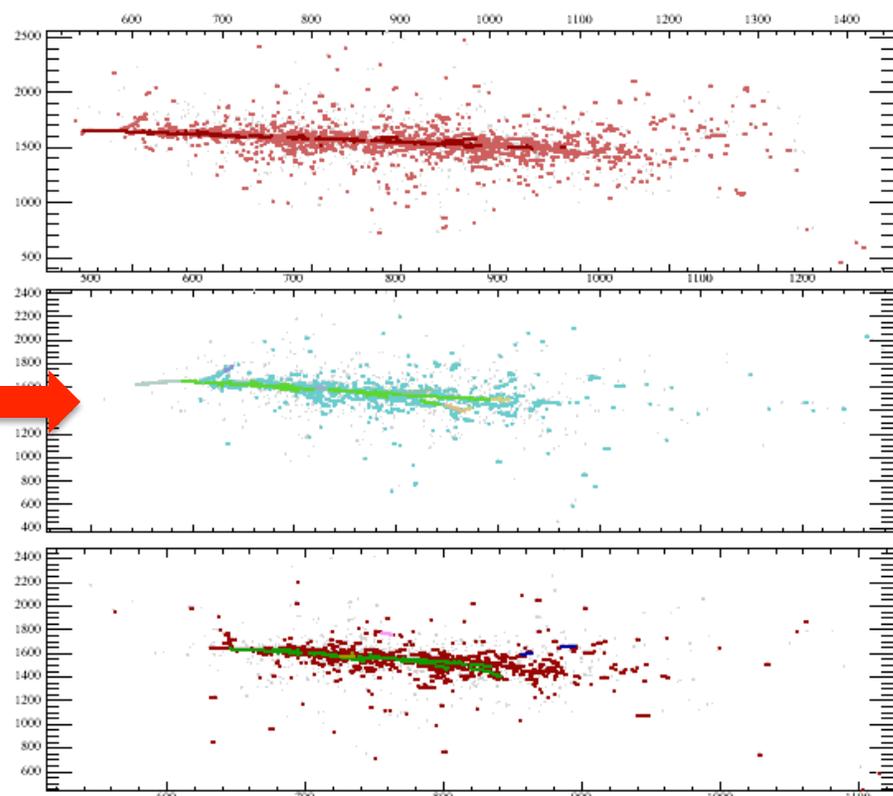
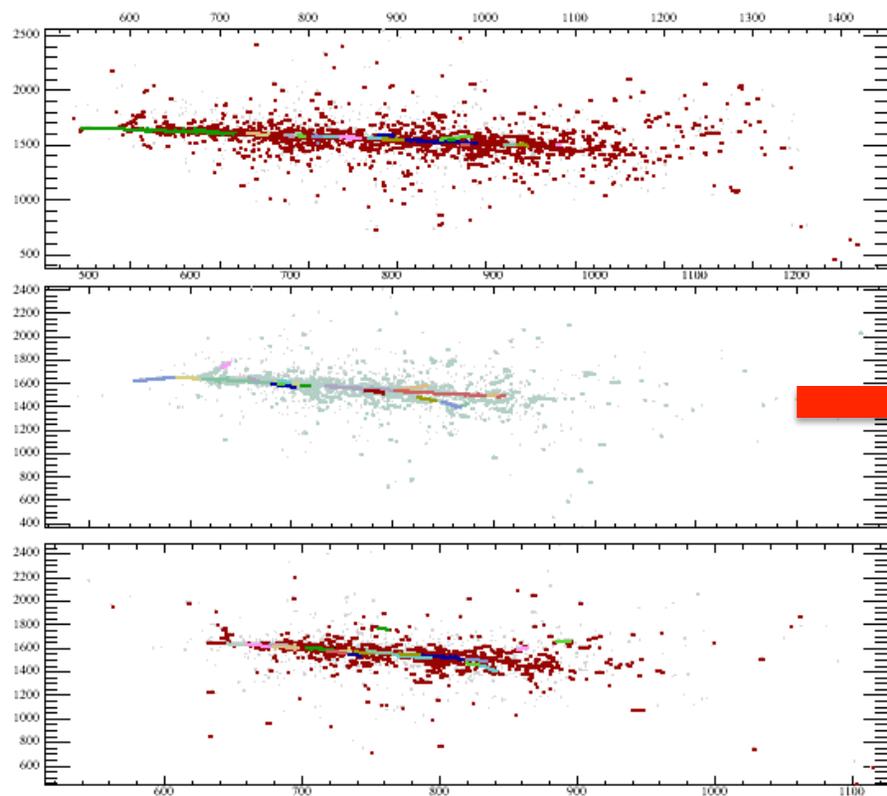
# Showers with Hough lines



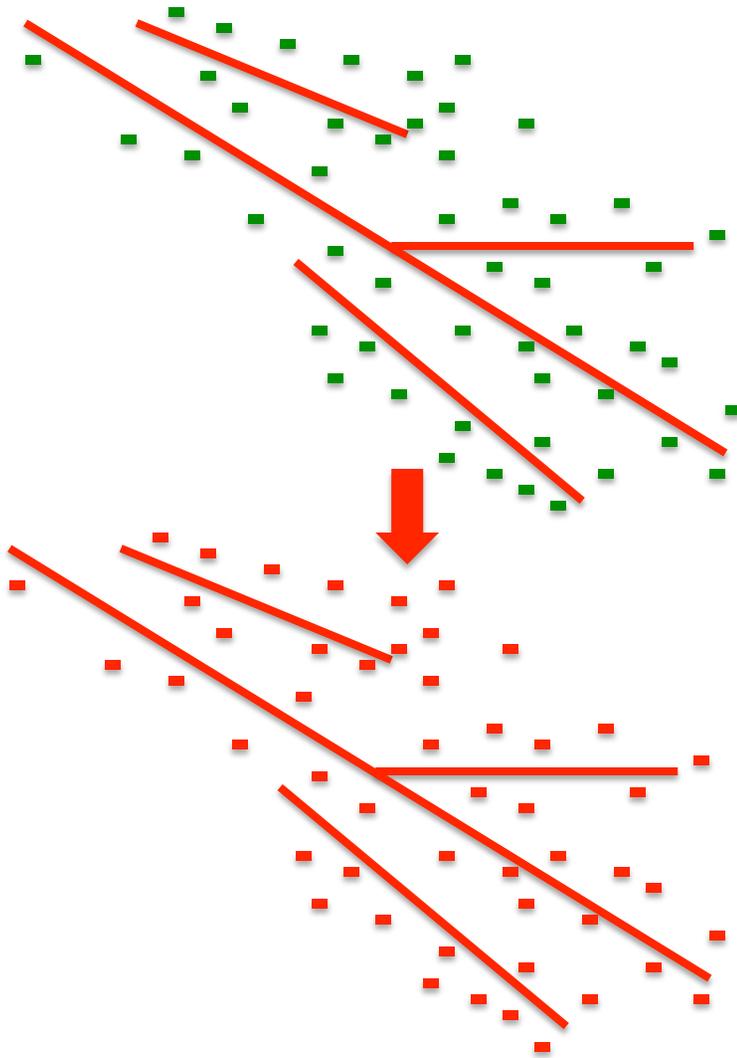
Hough line finder very effective at finding lines in showers

Now aiming to capitalize on this, merging Hough lines together to construct showers

Merge Hough lines using the distance between the line segments and angle between slopes  $< 20^\circ$



# The fuzzy cluster remnants

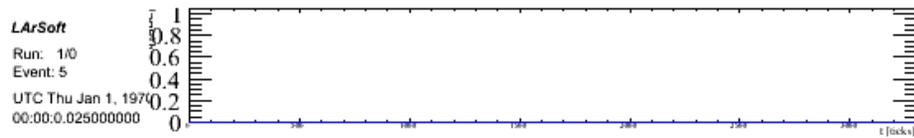
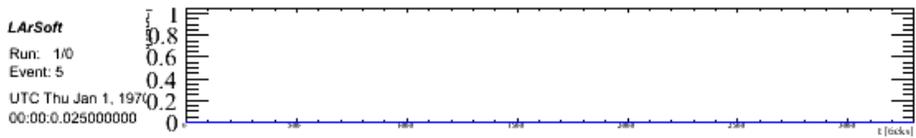
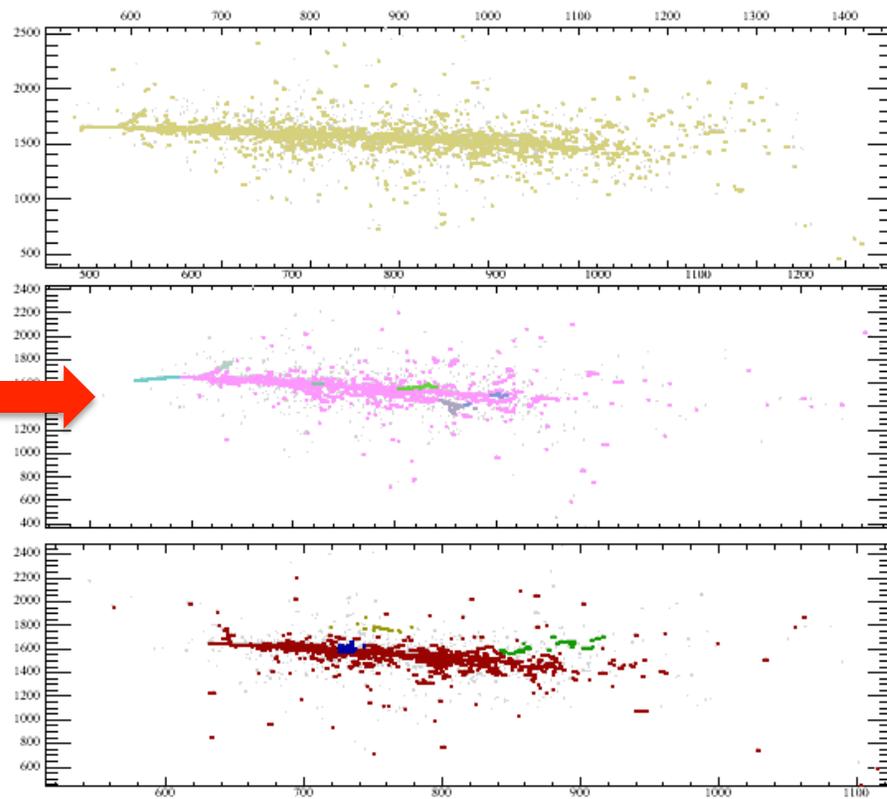
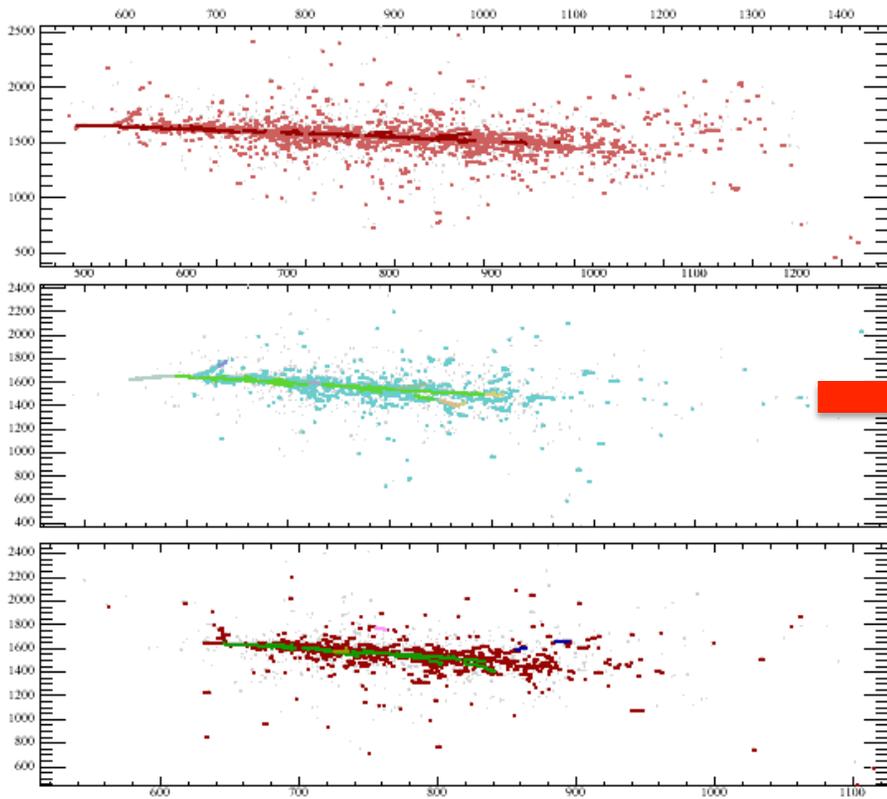


The issue remains with what to do with the left over fuzzy cluster points

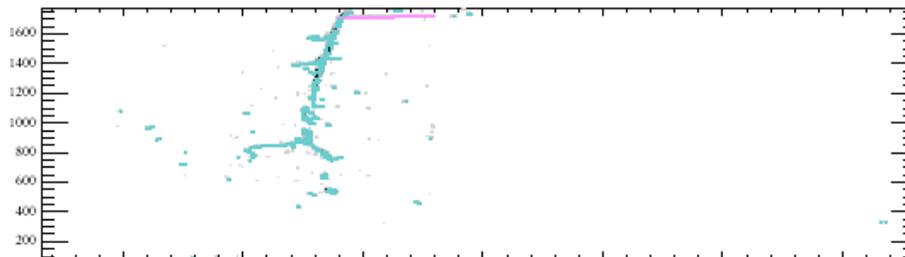
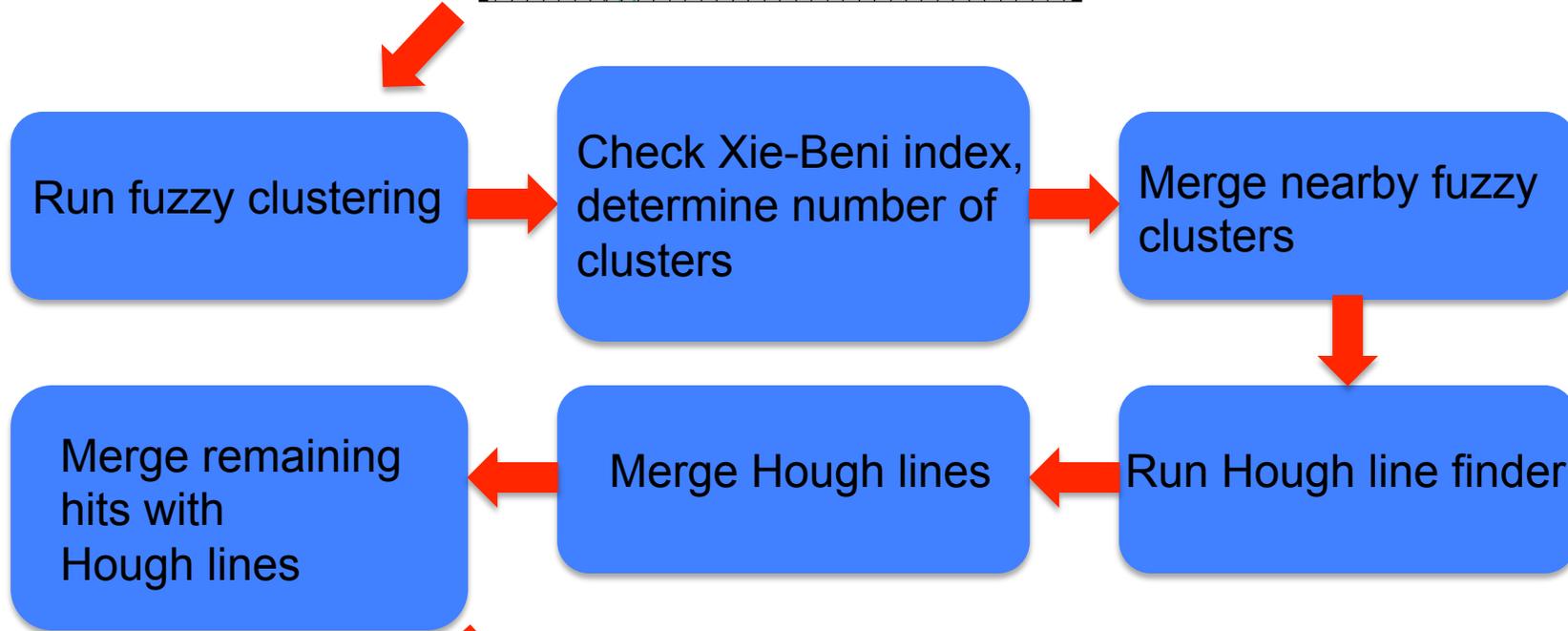
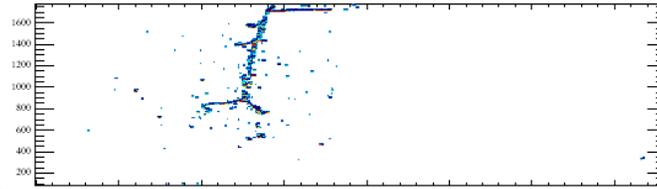
Currently merging these points into the nearest Hough line segment using:

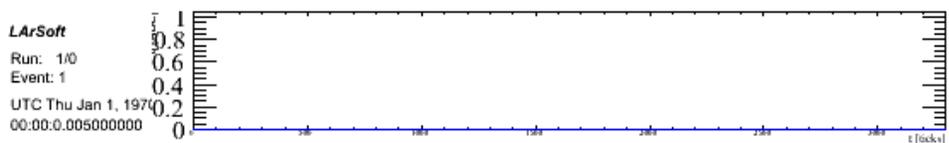
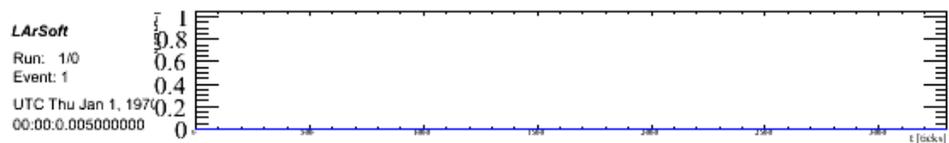
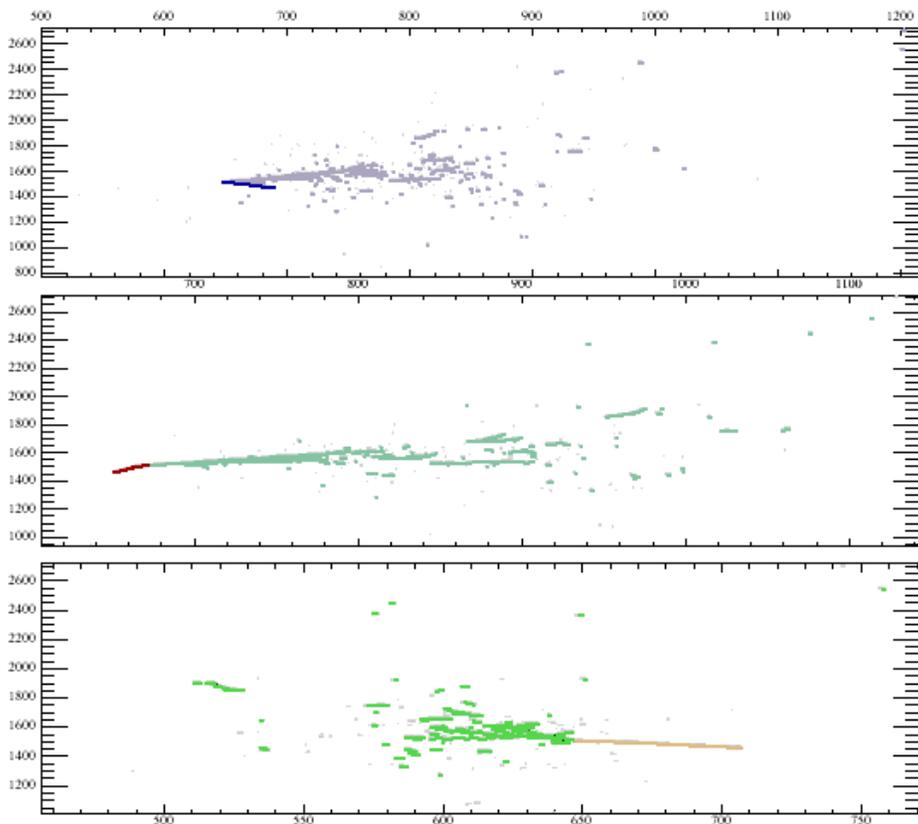
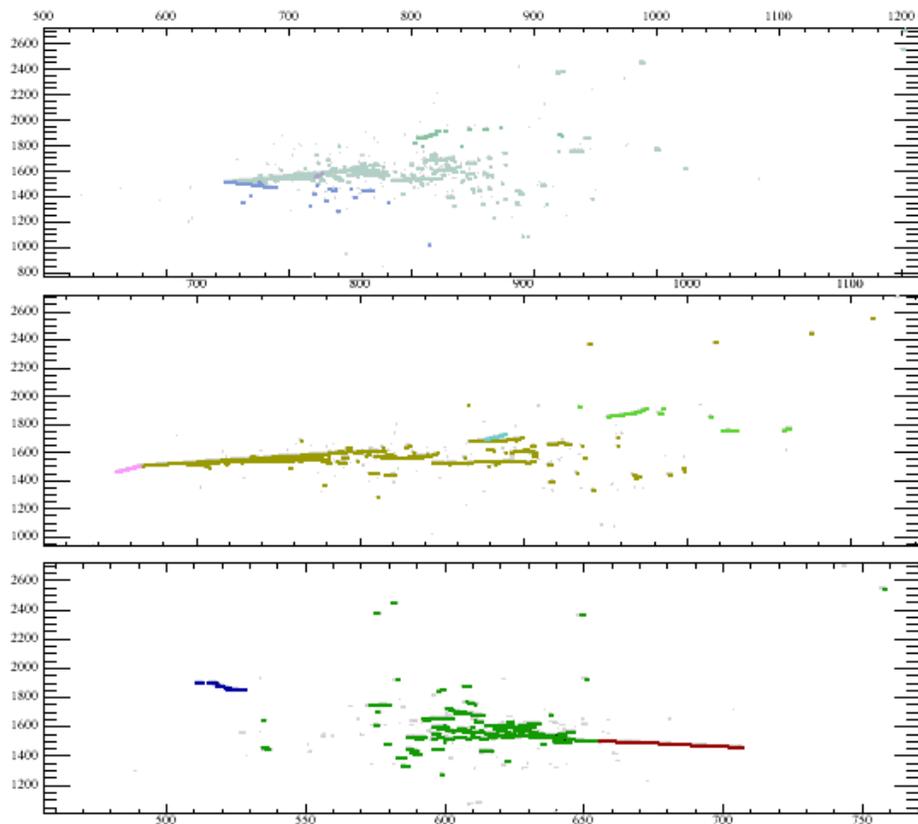
$$d_{\text{weighted}} = \frac{\text{distance point to line segment}}{\text{length of line segment}}$$

Goal is to give longer line segments more weight



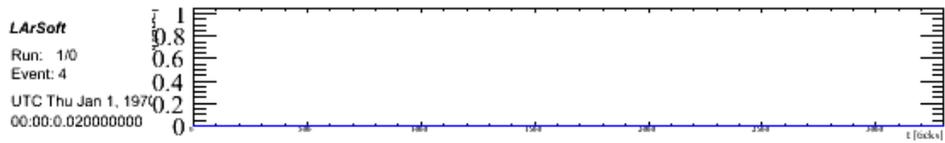
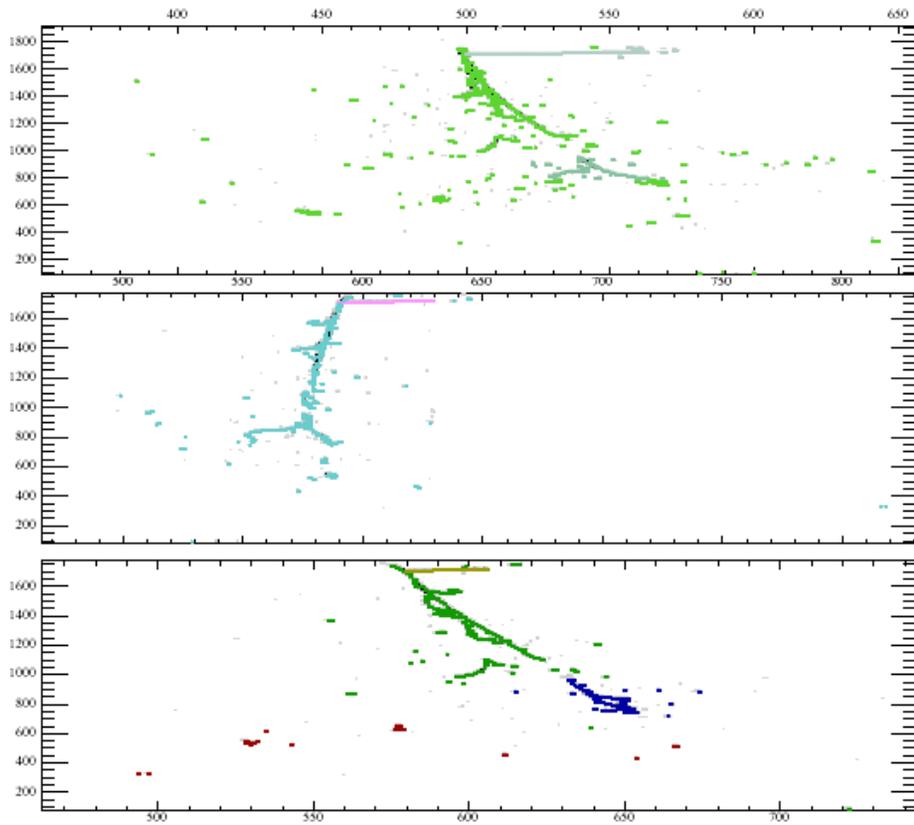
# Overview of the algorithm



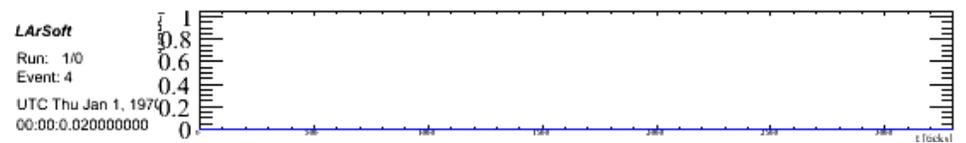
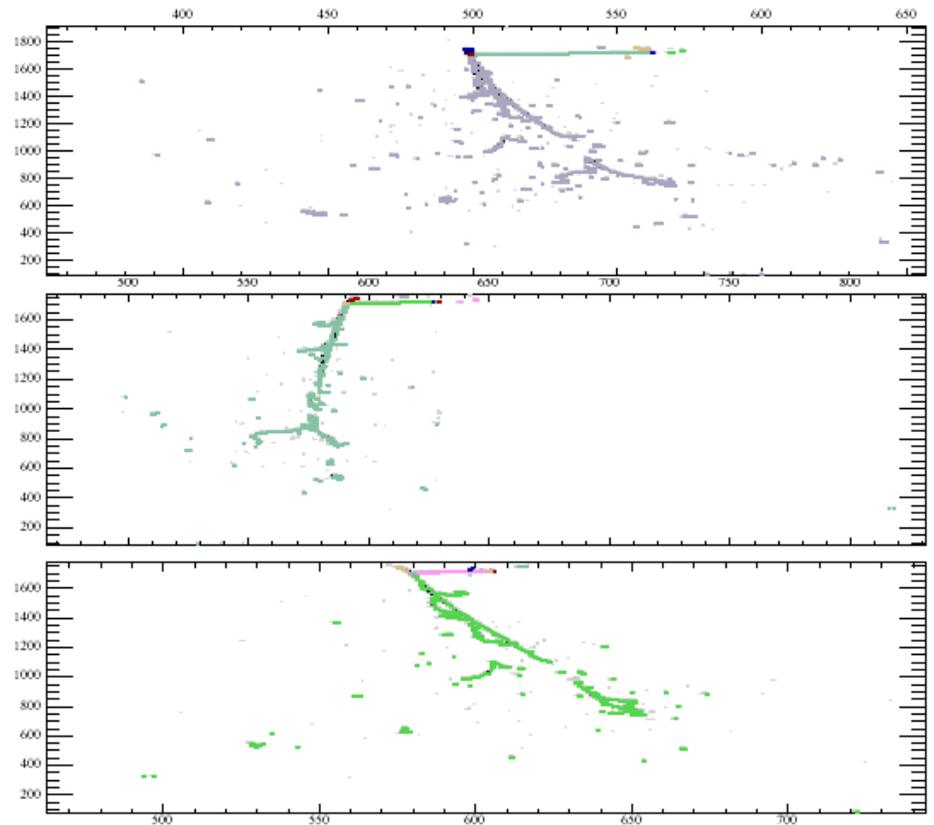


Fuzzy and Hough line clustering

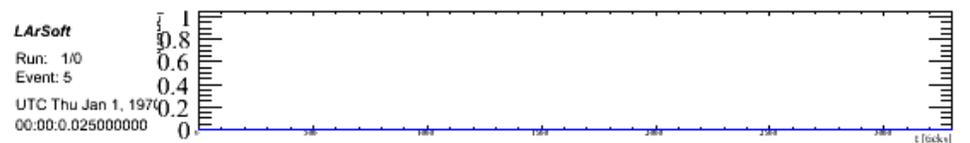
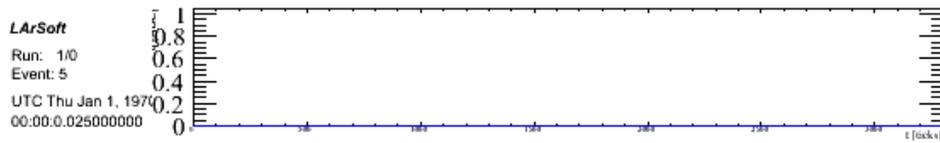
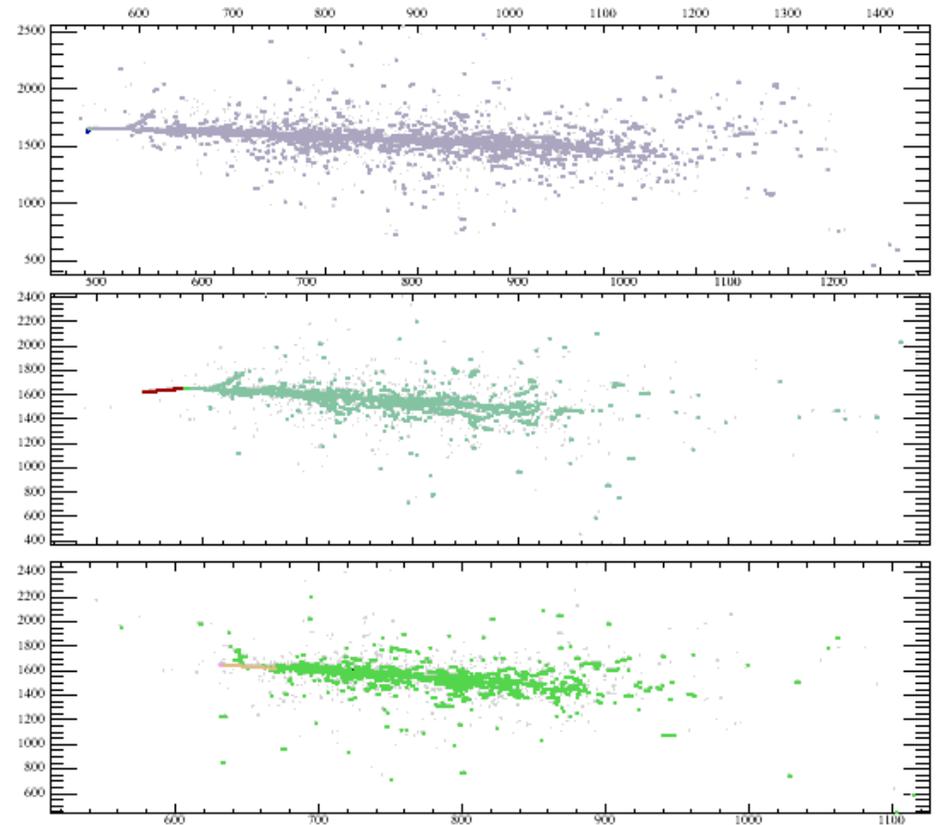
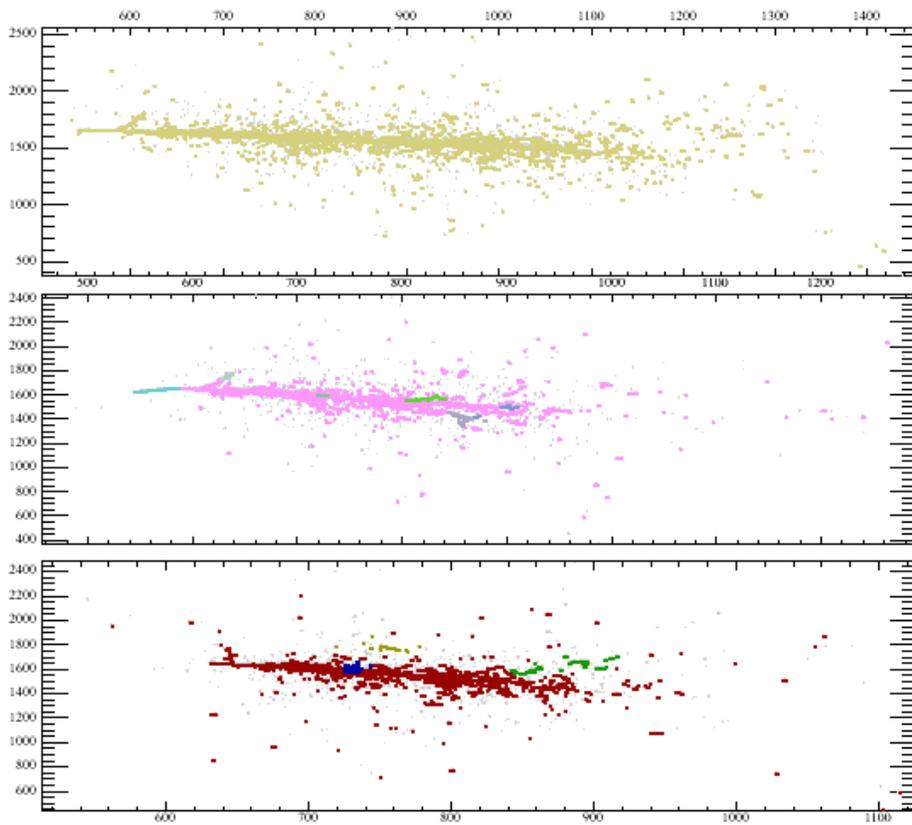
Cluster cheater



Fuzzy and Hough line clustering

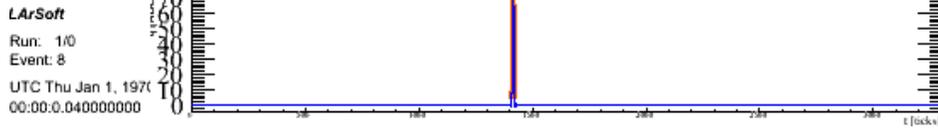
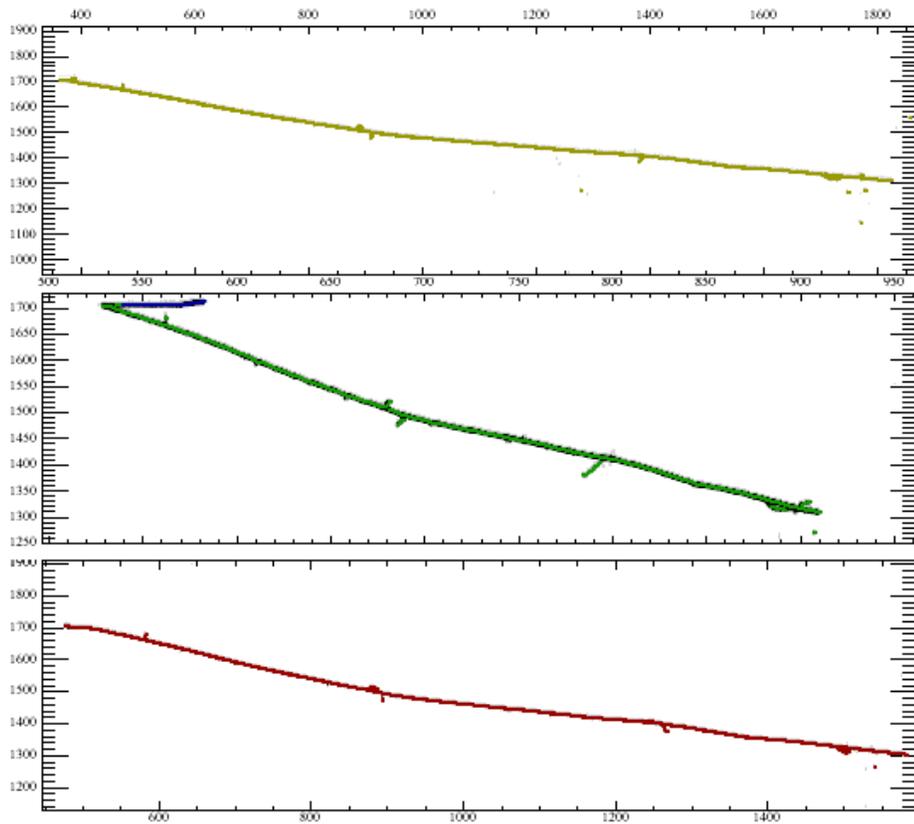


Cluster cheater

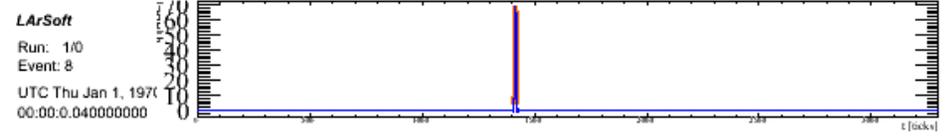
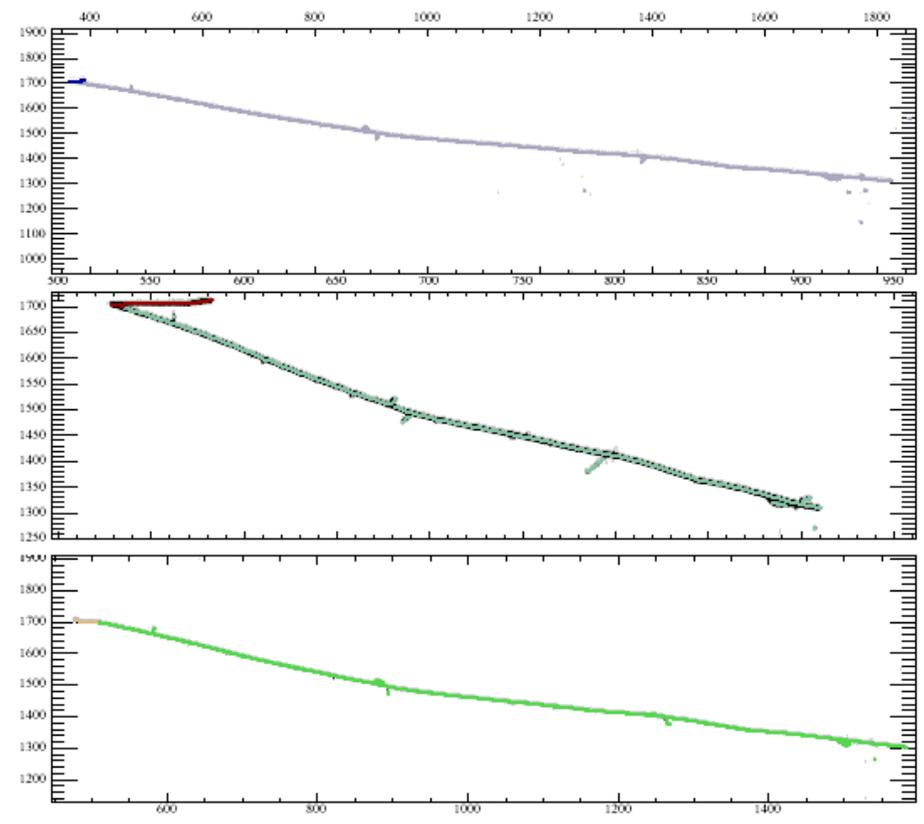


Fuzzy and Hough line clustering

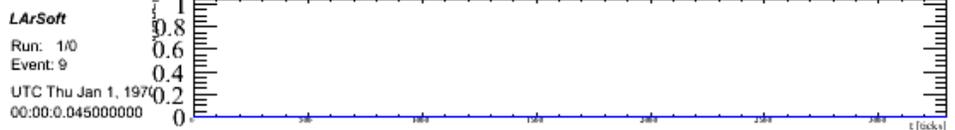
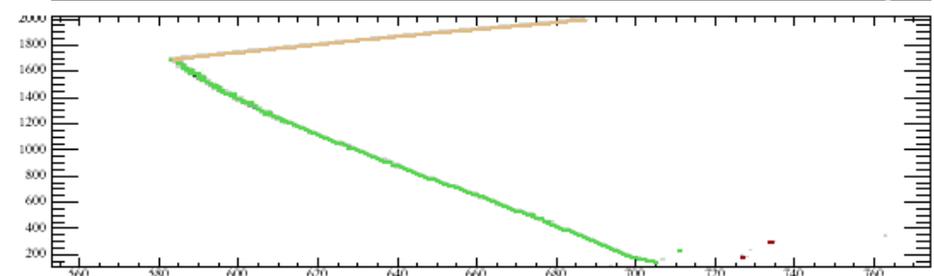
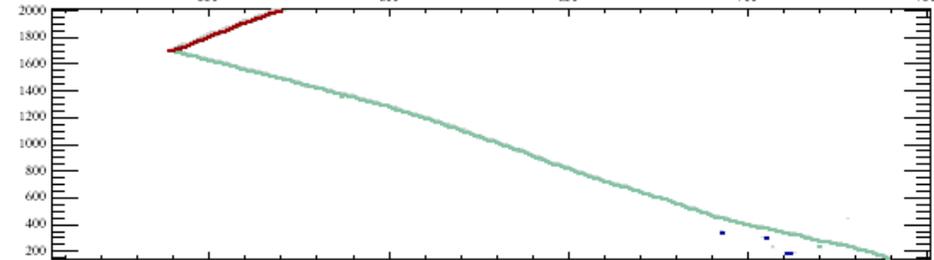
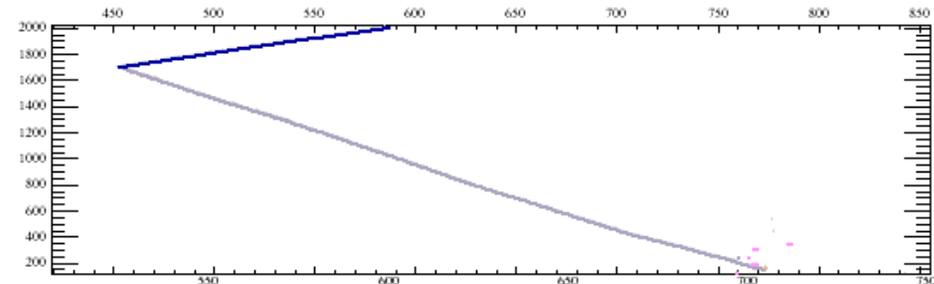
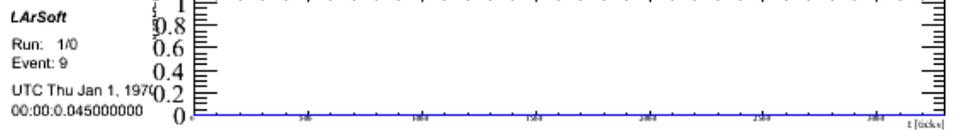
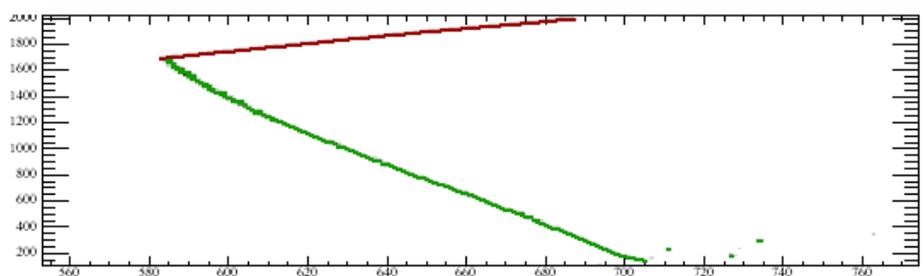
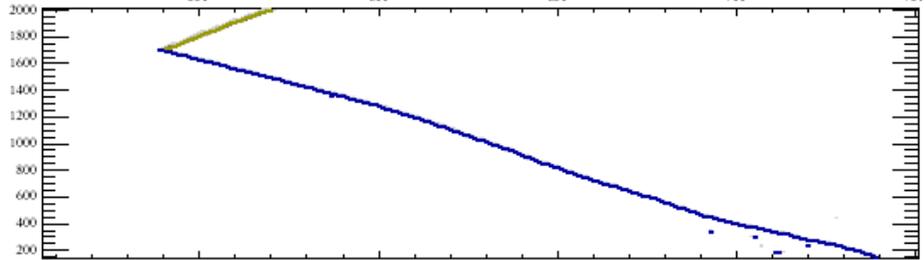
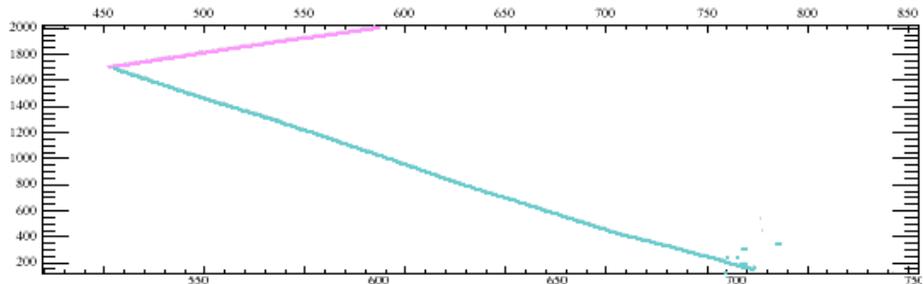
Cluster cheater



Fuzzy and Hough line clustering



Cluster cheater



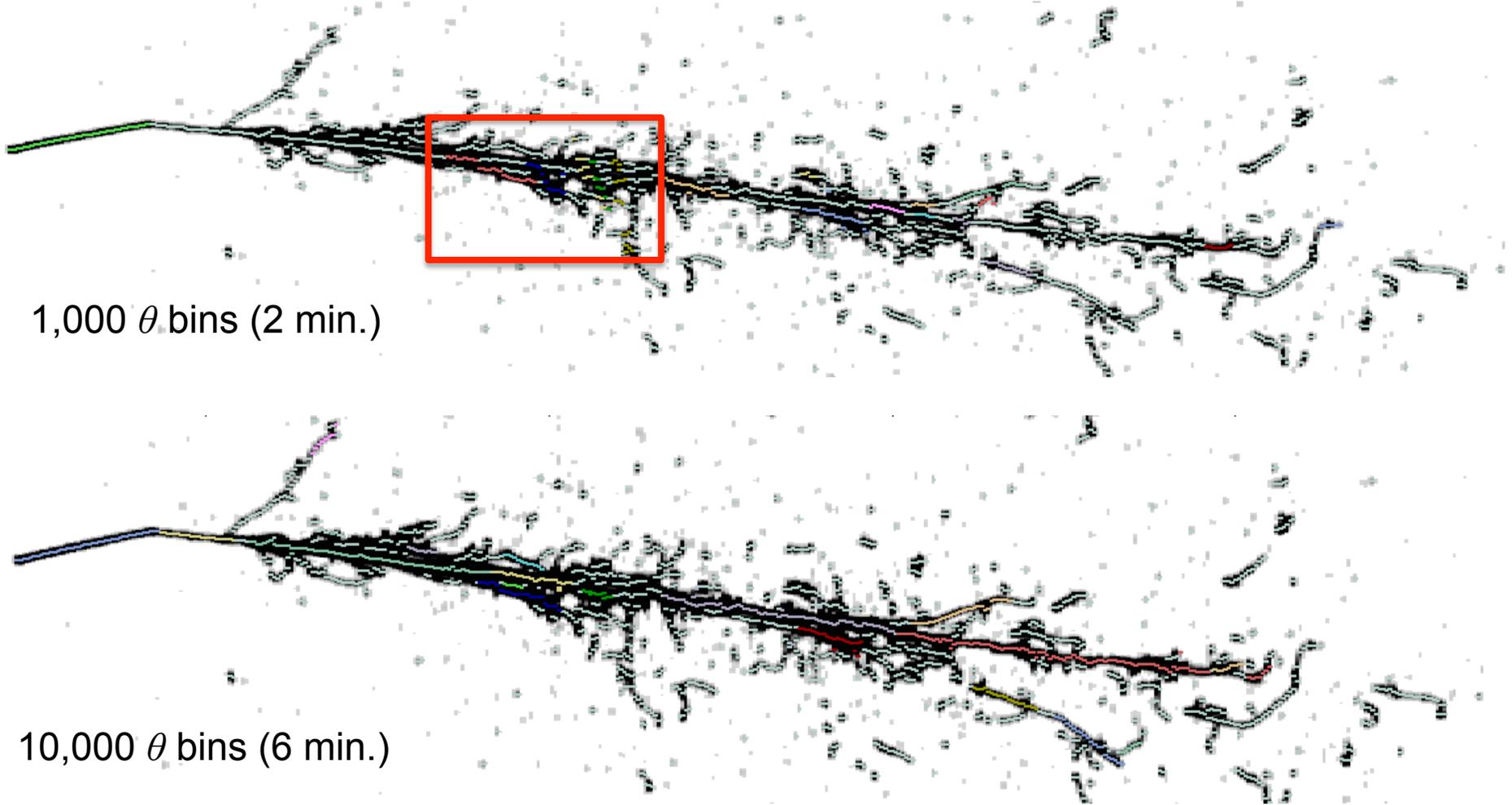
Fuzzy and Hough line clustering

Cluster cheater

# Hough transform performance

- Quality Hough lines are very dependent on resolution in the accumulator
- Low resolution (fast) has a higher fake rate than a higher resolution (slow)
- Ideally would run at low resolution, with fake veto

# Hough transform performance



1,000  $\theta$  bins (2 min.)

10,000  $\theta$  bins (6 min.)

Compare to 37 minutes (5,000  $\theta$  bins) for original Hough transform code

# Next steps

- Finish optimization
- Streamline code, might be too many unnecessary classes and functions
- Examine more events (only CCQE so far) to improve robustness
- Finalize the needed modules

Back up

# Where clustering was

- Want to finalize clustering occurring right after hit reconstruction
- Lots of tools already available (e.g. DBSCAN, Hough line finder, and End Point Finder)
- Goal is to bring them all together

# The basic algorithm

1. Initialize  $U=[u_{ij}]$  matrix,  $U^{(0)}$
2. At  $k$ -step: calculate the centers vectors  $C^{(k)}=[c_j]$  with  $U^{(k)}$

$$c_j = \frac{\sum_{i=1}^N u_{ij}^m \cdot x_i}{\sum_{i=1}^N u_{ij}^m}$$

3. Update  $U^{(k)}$ ,  $U^{(k+1)}$

$$u_{ij} = \frac{1}{\sum_{k=1}^C \left( \frac{\|x_i - c_j\|}{\|x_i - c_k\|} \right)^{\frac{2}{m-1}}}$$

4. If  $\|U^{(k+1)} - U^{(k)}\| < \epsilon$  then STOP; otherwise return to step 2.

The objective function we are trying to minimize:

$$J_m = \sum_{i=1}^N \sum_{j=1}^C u_{ij}^m \|x_i - c_j\|^2$$

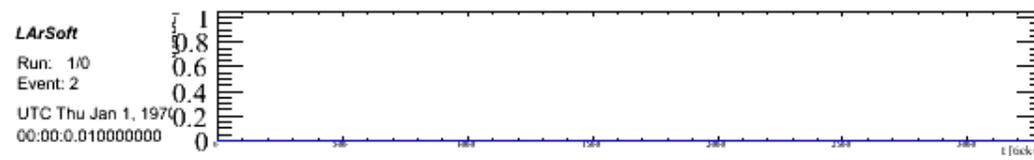
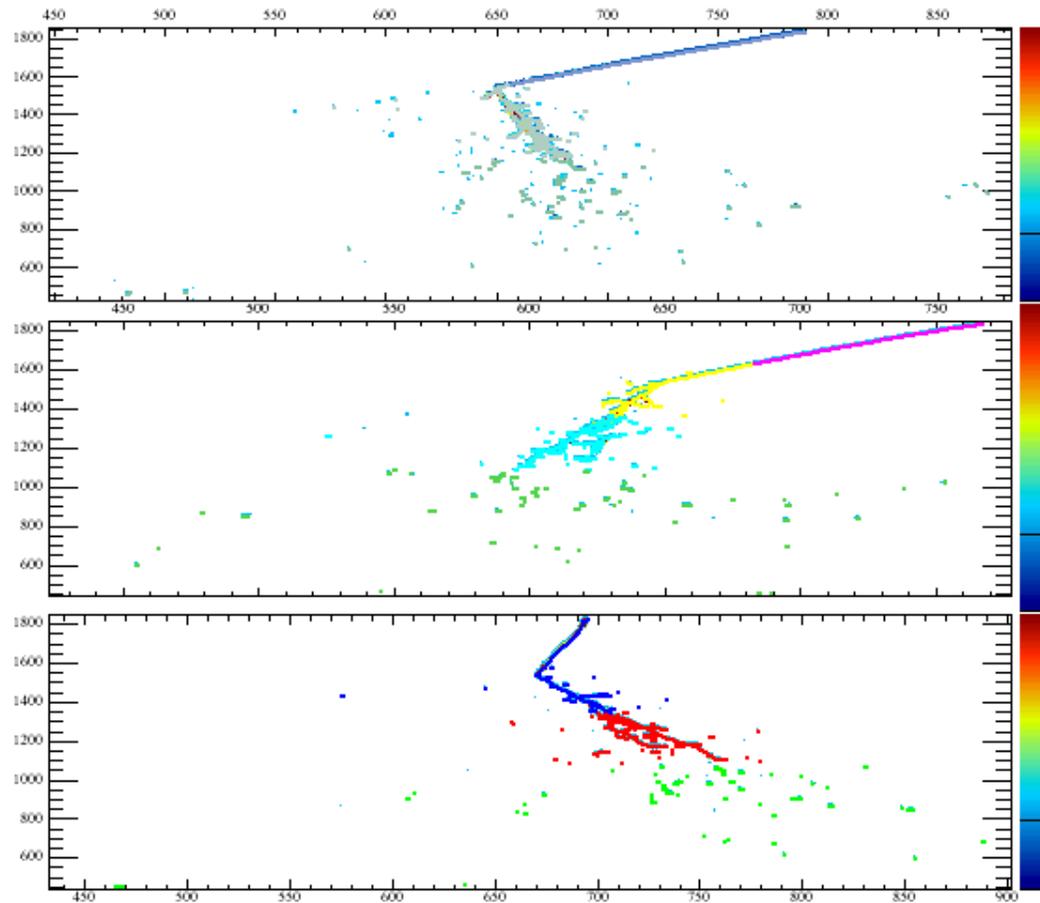
# Number of clusters

- The number of clusters needs to be given as an input
- The Xie-Beni index gives us a way to evaluate how well a certain cluster number works

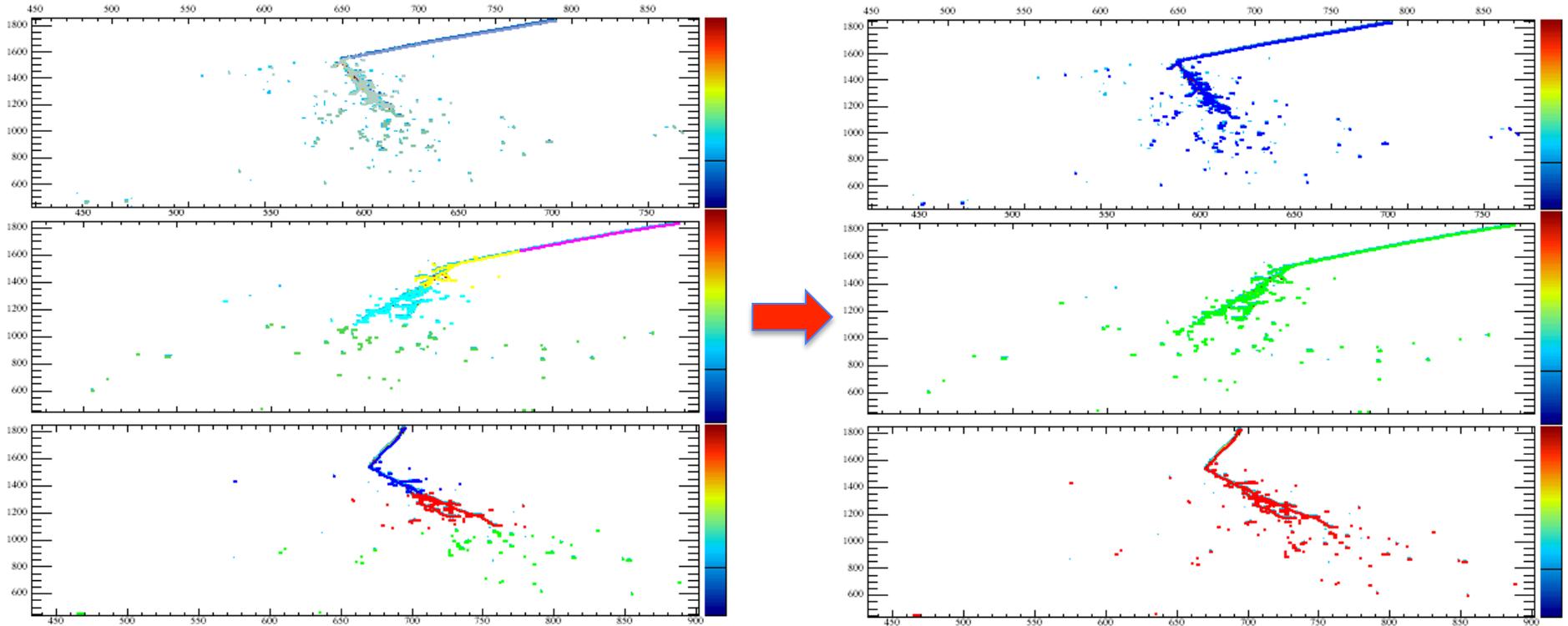
$$\frac{\frac{1}{n} \sum_{i=1}^n \sum_{k=1}^K u_{ik}^m \|x_i - c_k\|^2}{\min_{k,\ell} \|c_k - c_\ell\|^2}$$

Does not work with single cluster events by definition, we'll get to that later

There is a problem with the Xie-Beni Index though

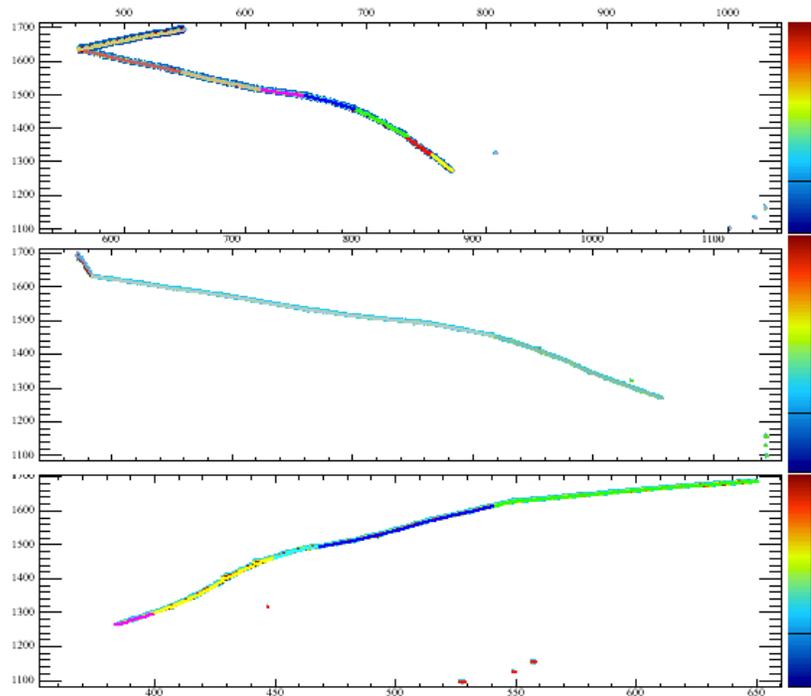


## Look to merge clusters using Euclidean distance



1. Look for point in cluster  $i$  closest to centroid in cluster  $j$
2. Compare that point to the point in the cluster  $j$  closest to centroid of cluster  $i$
3. Merge the clusters

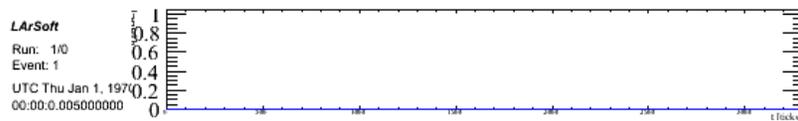
Now look for Hough lines and expand them into clusters



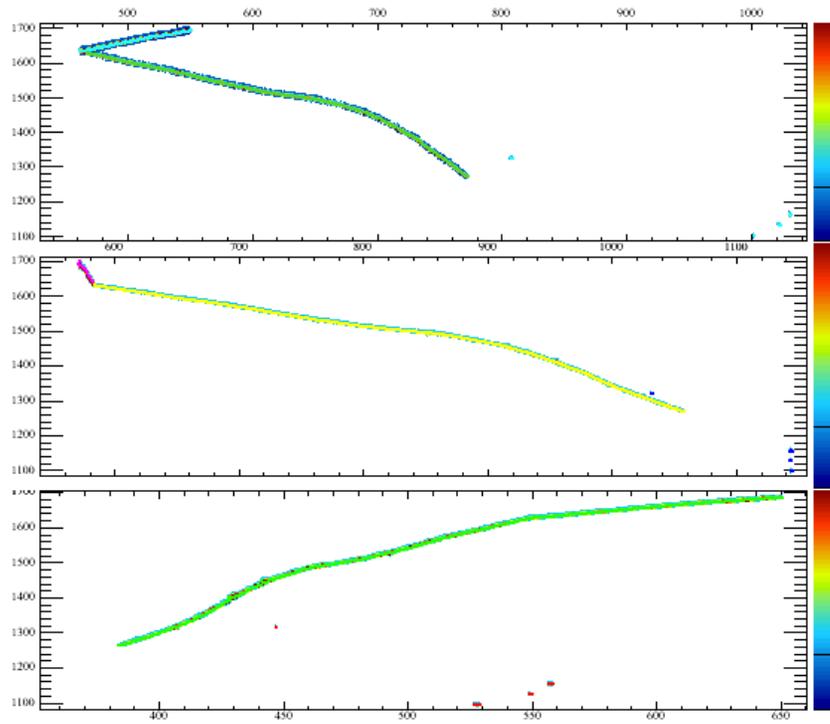
Search for Hough lines in the already identified fuzzy clusters

Hough line finder was sped up to make this practical

An issue remains with needing to merge the Hough lines though



## Merging the Hough lines



Pick a line and check at its end points

If a nearby line is found, check the angle between the slopes

If the angle between the slopes is  $< 30^\circ$ , merge the lines

