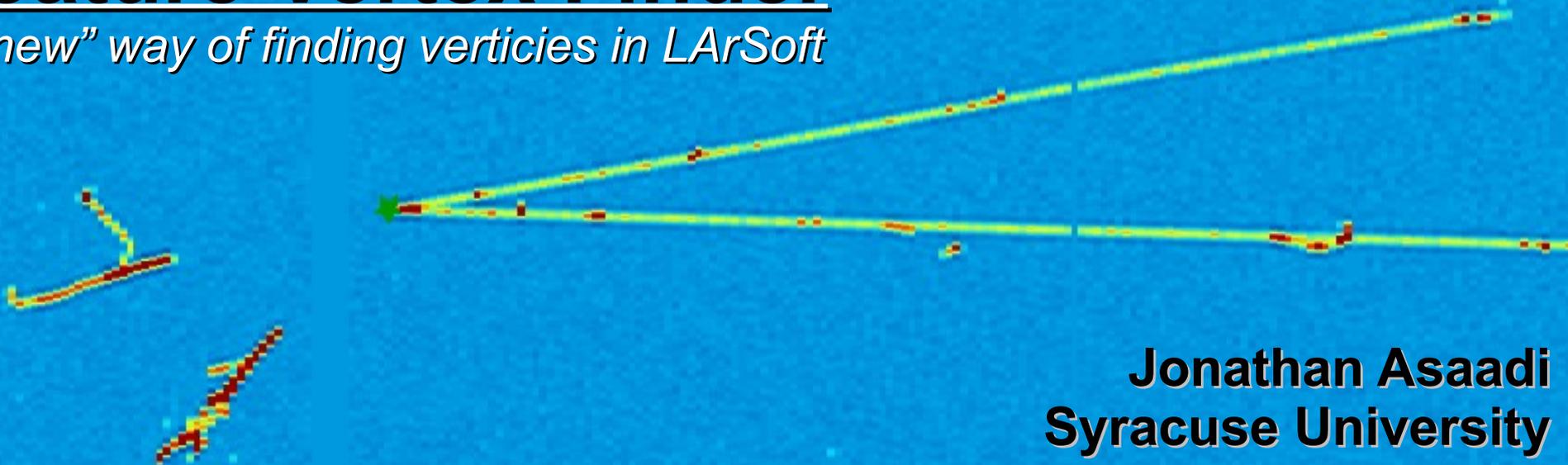


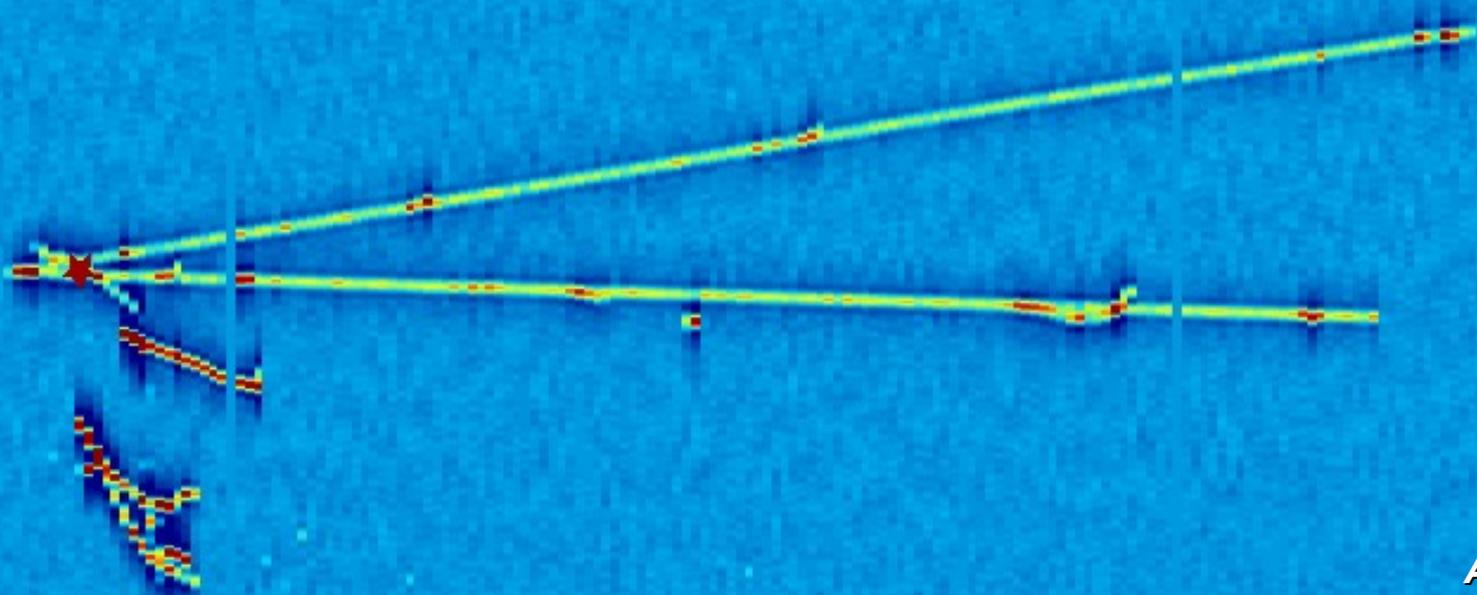
Feature Vertex Finder

A "new" way of finding vertices in LArSoft



Jonathan Asaadi
Syracuse University
LArSoft Meeting

5/19/13



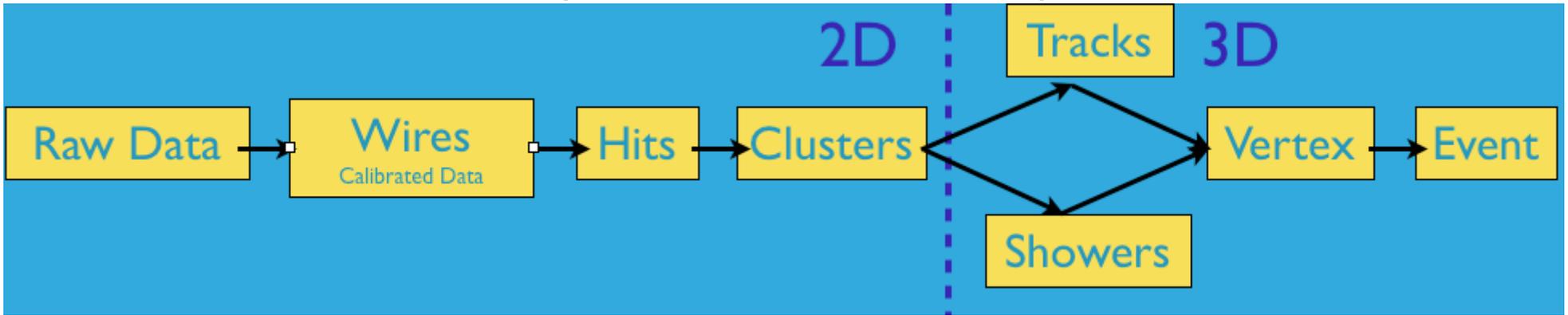
ArgoNeuT MC

Outline

- **Vertex finding in the broader reconstruction scheme**
 - How we do it now
 - How I imagine it could be
- **Proposal to change RecoBase Vertex object**
 - Adding vertex “strength” to the Vertex object
- **FeatureVertexFinder** (*I don't do well naming things*)
 - How it works (a.k.a building on the hard work of others)
 - Cluster Finding
 - Corner Finding
 - 2d / 3d matching
 - Preliminary performance plots (*still lots of improvements to be made*)
 - Some plots using vertex strength to show performance
- **Backup Slides**
(more details about the algorithm than can fit in one talk)

Reconstruction

(How we do it now)



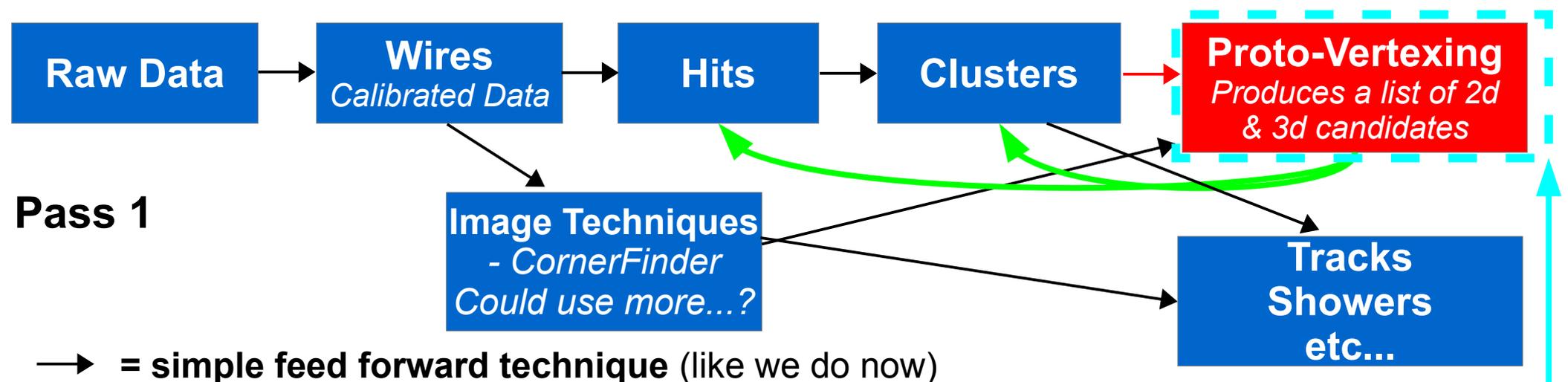
(Part of what is described below is a cheat and an over-simplification...but is more used to illustrate my point that if we use information differently we can improve our reconstruction)

Right now, in LArSoft, we follow a very linear chain of reconstruction with one module calling upon a previous module (or maybe a few previous modules) to reconstruct the next object that gets passed up the chain

An event vertex is reconstructed very late in the chain (mostly taking in 3d objects) and is seldom (if ever) used to inform other reconstruction

Reconstruction

(How I imagine it could be)



Pass 1

→ = simple feed forward technique (like we do now)

→ = new link in the chain (i.e. FeatureVertexFinder)

→ = feed back to previous algorithms

■ = Existing algorithm

■ = New algorithm

I haven't done this yet...just thinking out loud

Most of this talk will focus on this new module

During our first pass we make simple objects based on available information. However, since many of these object made during the first pass can inform/improve reconstruction algorithms, we identify those links and re-run reconstruction using this information

Better Hits

(e.g. Hits near a vertex may need to be handled differently)

Refine Clusters

(e.g. Clusters that know where the origin may be as input)

Change to RecoBase Vertex

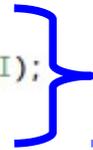
As of right now the only 3d vertex class in LArSoft only has XYZ and ID as members of its class

I would like to add “strength” to this class

→ I could see this having the vertex strength defined for each module that generates 3d vertices (would be up to the author to tell everyone the scale and what it means)

→ For FeatureVertexFinder I would use the same definition I outlined later based on matches and merges (+ tweaks & suggestions from others)

```
namespace recob {  
  
class Vertex {  
public:  
    Vertex(); // Default constructor  
  
private:  
  
    double fXYZ[3];    ///< location of vertex  
    int    fID;        ///< id number for vertex  
  
#ifndef __GCCXML__  
public:  
  
    explicit Vertex(double *xyz,  
                    int    id=util::kBogusI);  
    void      XYZ(double *xyz) const;  
    const int ID() const;
```



double strength() const;

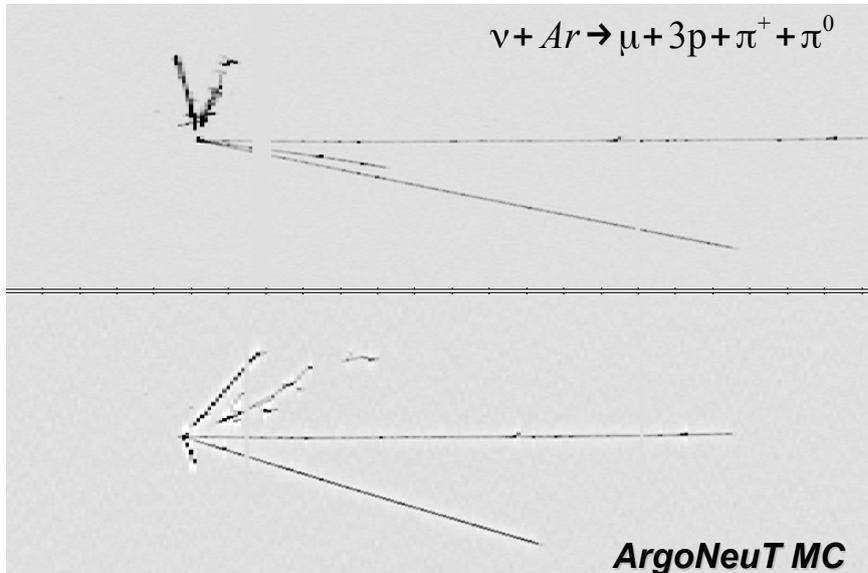
Can I have permission to Modify RecoBase/Vertex ?

Anything else we should add?

Feature Vertex Finder

Easiest method is to show how module works is to step through an example MC event

Step 1: Start with a wire information



Step 2: Run the event through a generic reconstruction chain up to 2-d clusters (*keep CalWire*)

HitFinder (GausHitFinder)

2-d Cluster (dbCluster)

2-d LineFinding (HoughLineFinder)

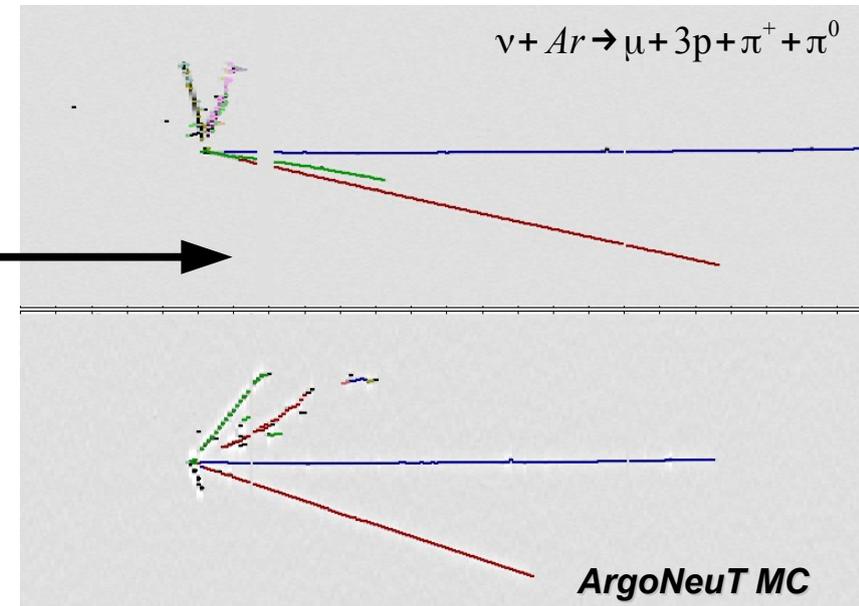
2-d LineFinding (LineMerger)

Note:
These are
just 2-d
Cluster
modules

What I am showing here are reconstructed hits and the 2d-Clusters from LineMerger

- In principal you could have used any 2-d cluster module
- I chose linemerger because it seemed to give me sensible clusters for finding a vertex

At this point in the reco chain I run FeatureVertexFinder



Feature Vertex Finder

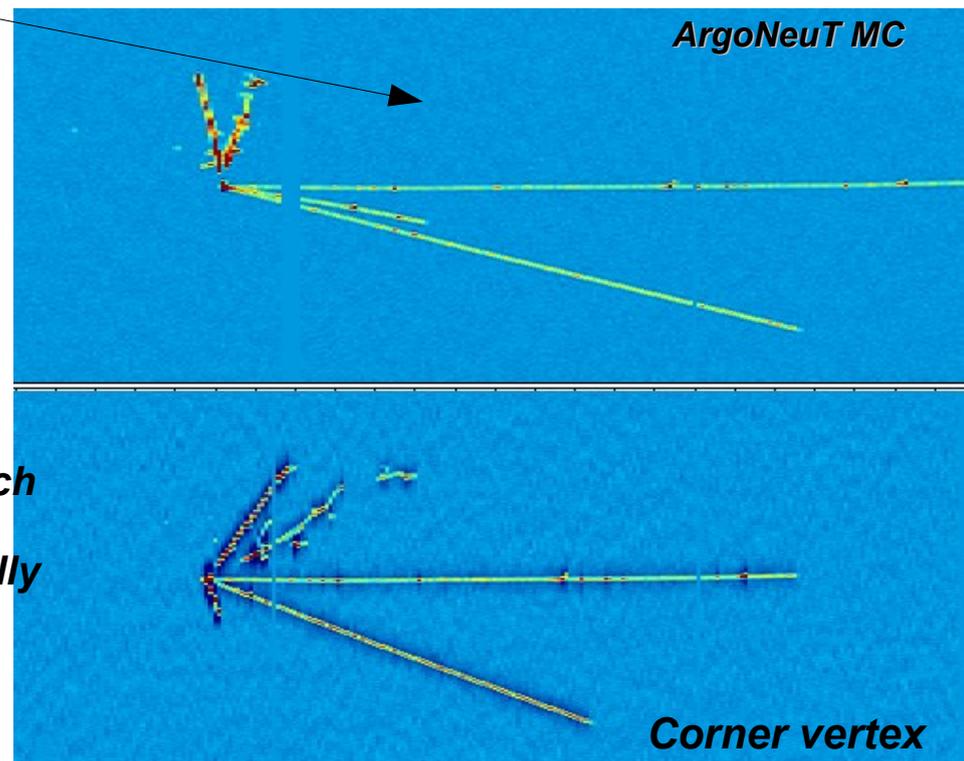
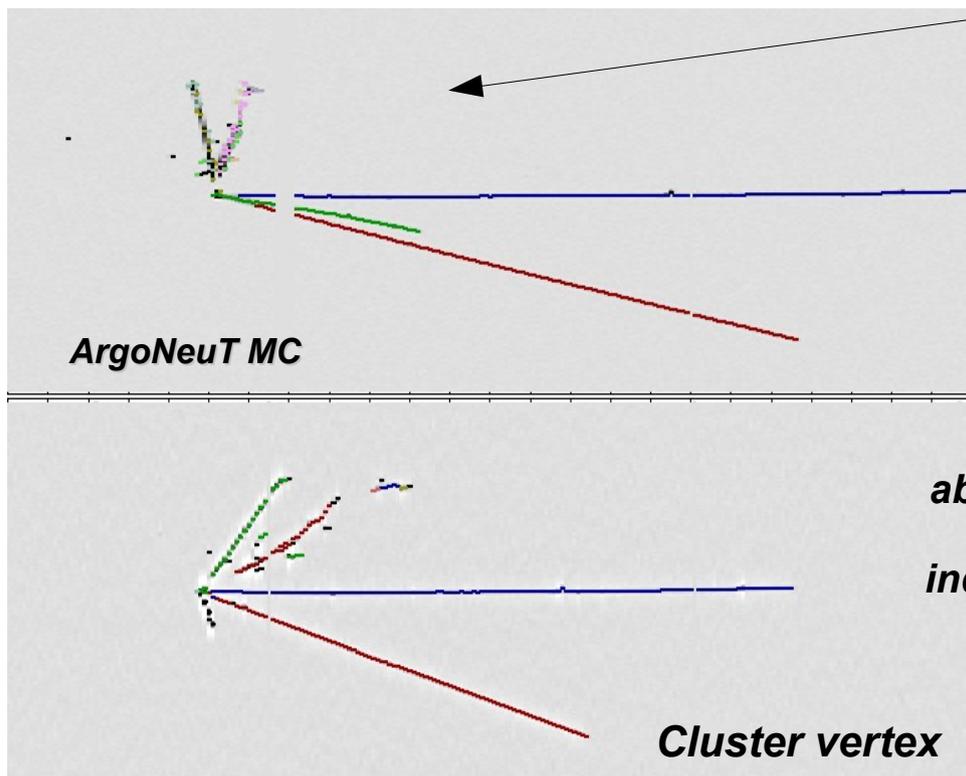
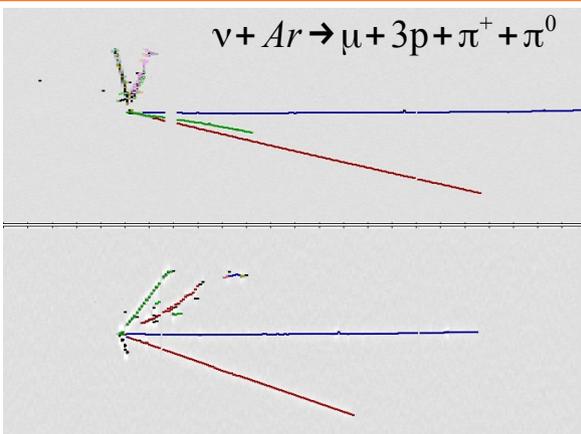
At this point FeatureVertexFinder looks at the event in two ways

(Method 1) 2d/3d Cluster Vertex

- Using the 2d-cluster information and calculating slopes and intercepts in 2d and then matching between planes to form 3D candidates

(Method 2) 2d/3d Corner Vertex

- 2d "Corner" finding using ConerFindingAlg and matching Between planes to form 3d candidates

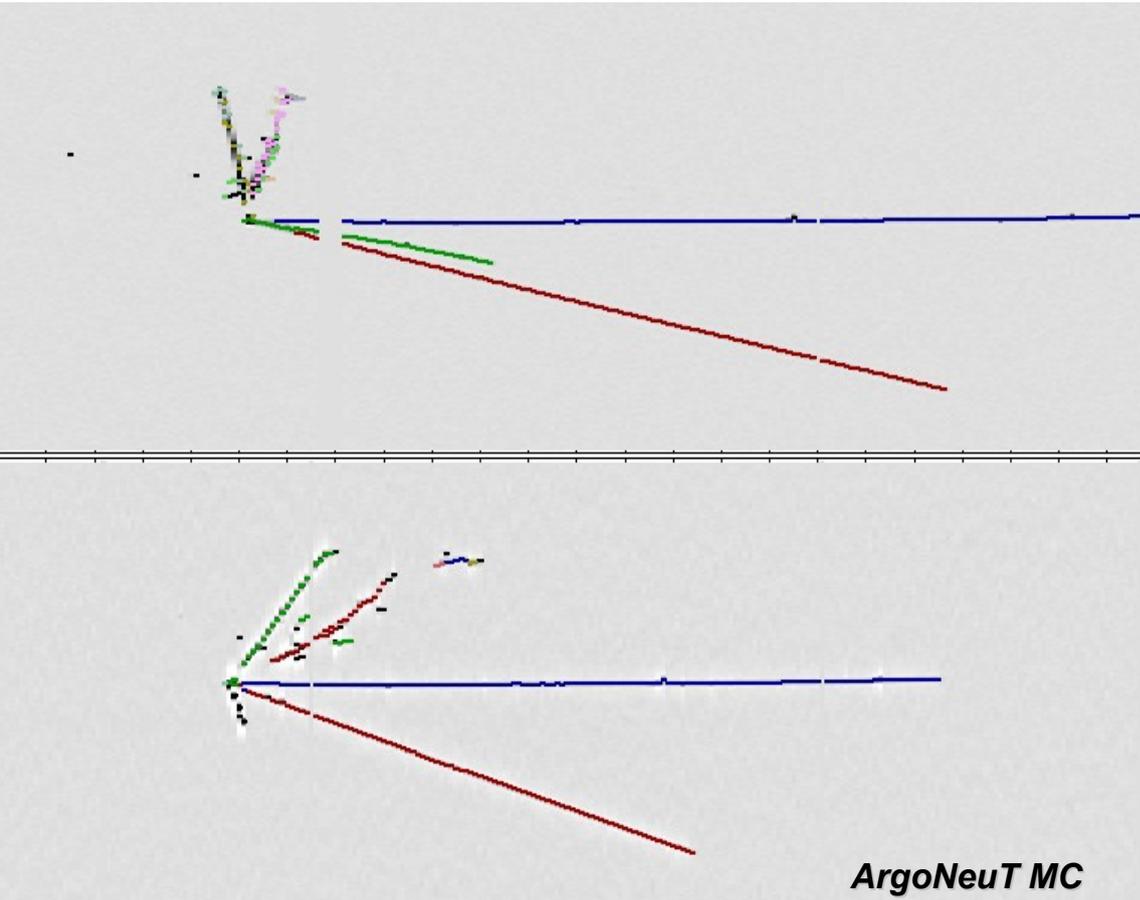


*I'll talk
about each
way
individually*

Feature Vertex Finder

2-d Slopes

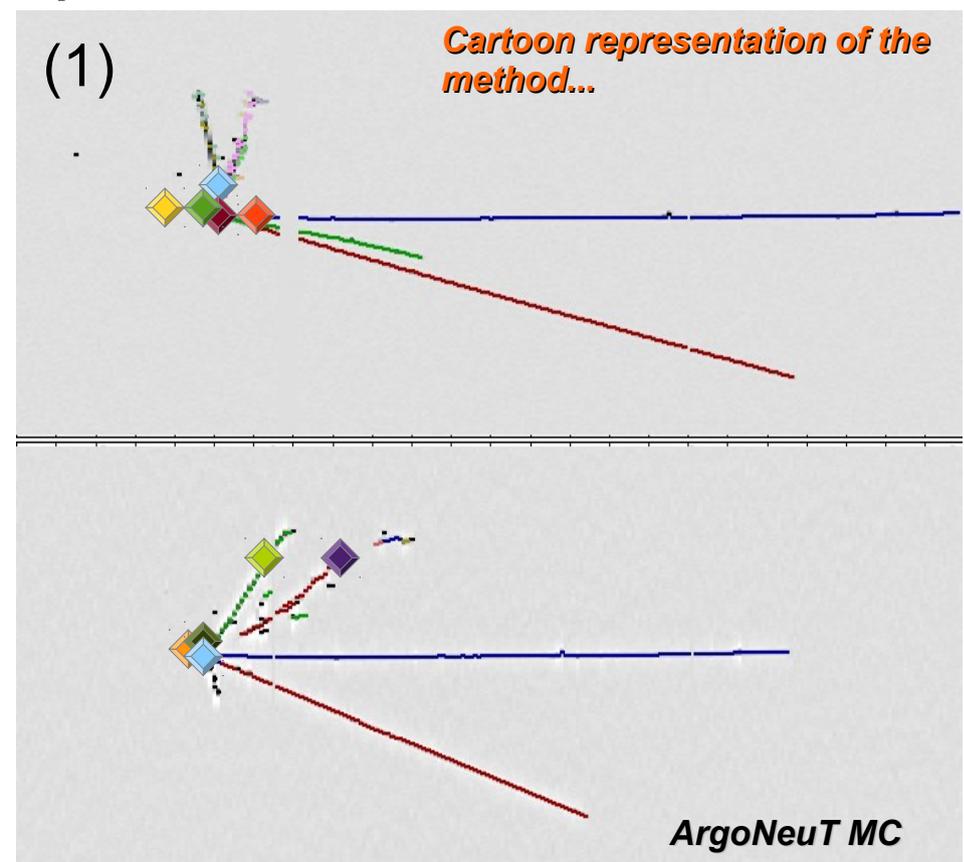
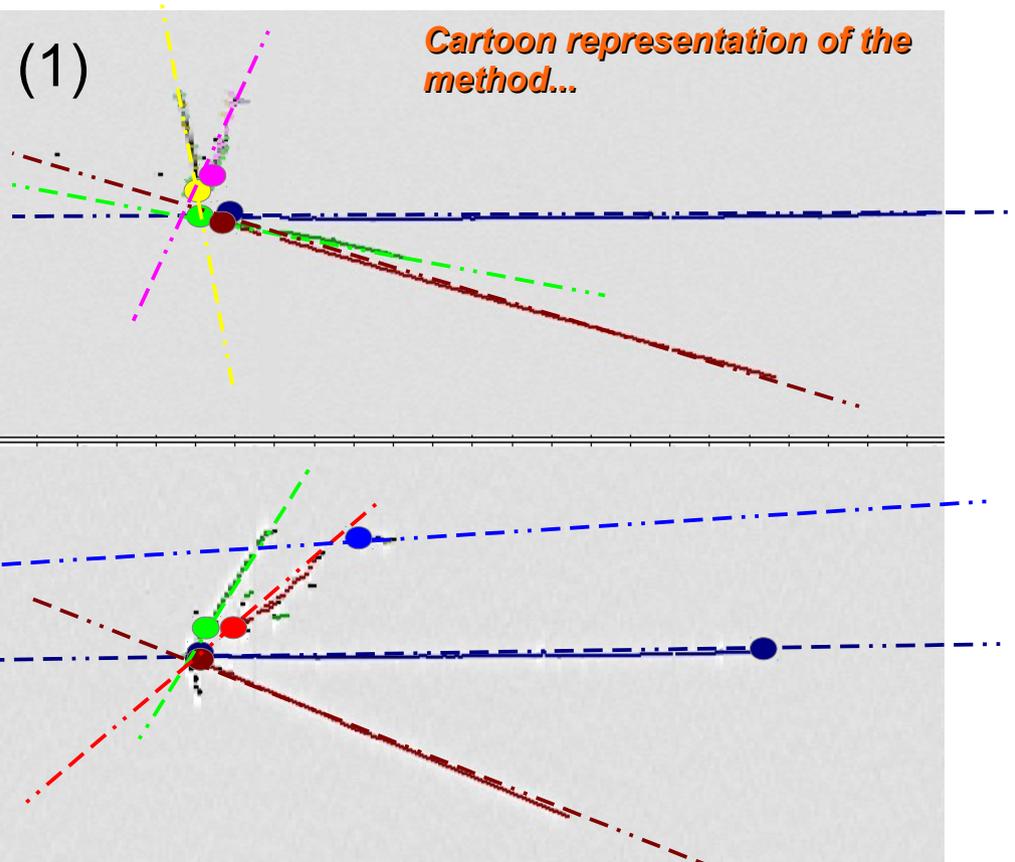
- For every cluster I take the hits for that cluster and fit a 1st order polynomial to the hits (pol1) and save the slope
- I use a fit instead of the reco cluster slope ($dT/dW = \text{delta Time} / \text{delta Wire}$) because not every cluster module calculates this
- I also save the start and end position (wire and time) for each cluster
- I do this since I expect the clustering algorithms (at this point) to get the actual start and end position switched sometimes



Now I search for **2-d cluster vertex candidates** in each plane

Feature Vertex Finder

2-d Slopes



(1) Using the slope and the start point of each cluster I calculate all the intersection points in each plane between all the lines
(I also repeat this procedure using the end points)

Greater details in the backup slides...

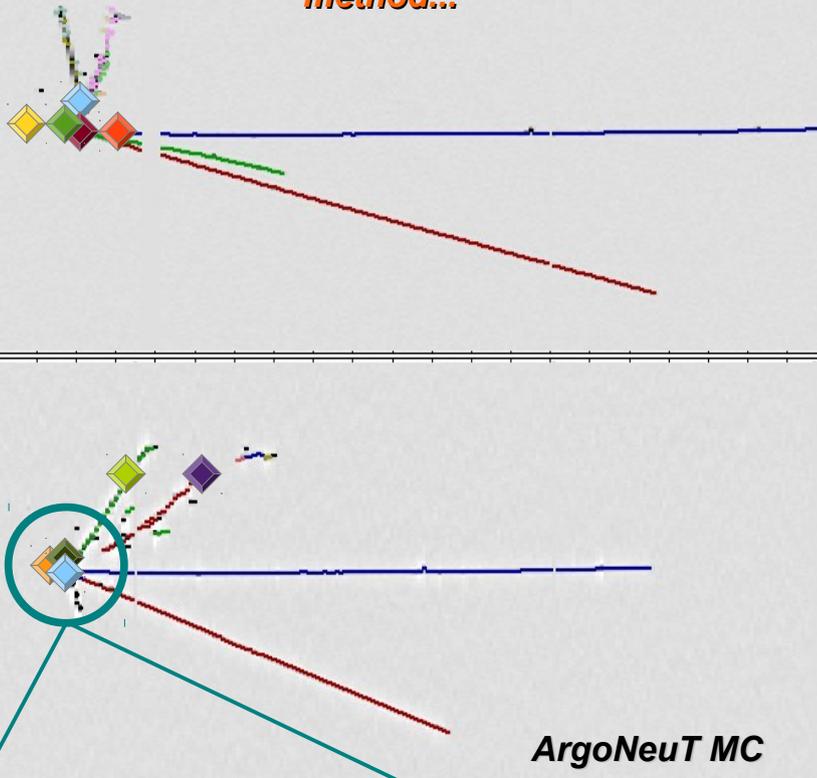
Now with a long list of 2-d cluster vertex candidates (wire & time) in each plane I look for **3-d cluster vertex candidates**

Feature Vertex Finder

Matching between views

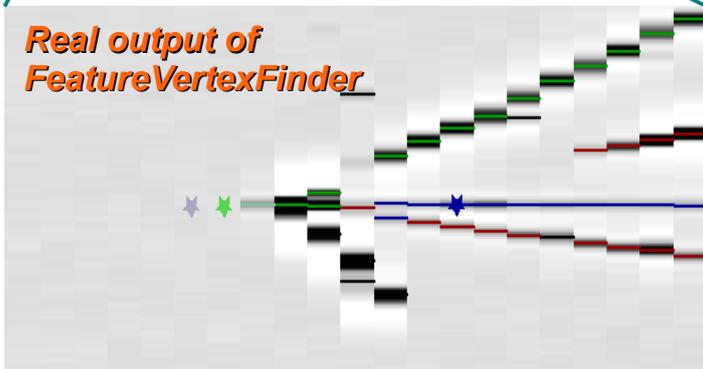
(2)

Cartoon representation of the method...



Zooming in on one view

Real output of FeatureVertexFinder



(2) Loop over each 2-d cluster vertex candidates ($channel_1$ and $time_1$) and look to see if there is a corresponding cluster vertex candidate ($channel_2$ and $time_2$) in a different plane and see if the channels intersect ($ChannelsIntersect(C1, C2, Y, Z)$). This gives me a Y and Z coordinate

Finally I require that the time difference between the two vertex candidates ($abs(time_1 - time_2)$) is within 1.5 times the expected offset between planes ($TimeOffsetV, TimeOffsetU, TimeOffsetZ$)

If the vertex satisfies both these conditions ($ChannelsIntersect$ and $1.5TimeOffset$), then calculate the vertex X coordinate ($TickstoX$)

But we aren't done yet!

So now I have a list of **2d/3d cluster vertices**

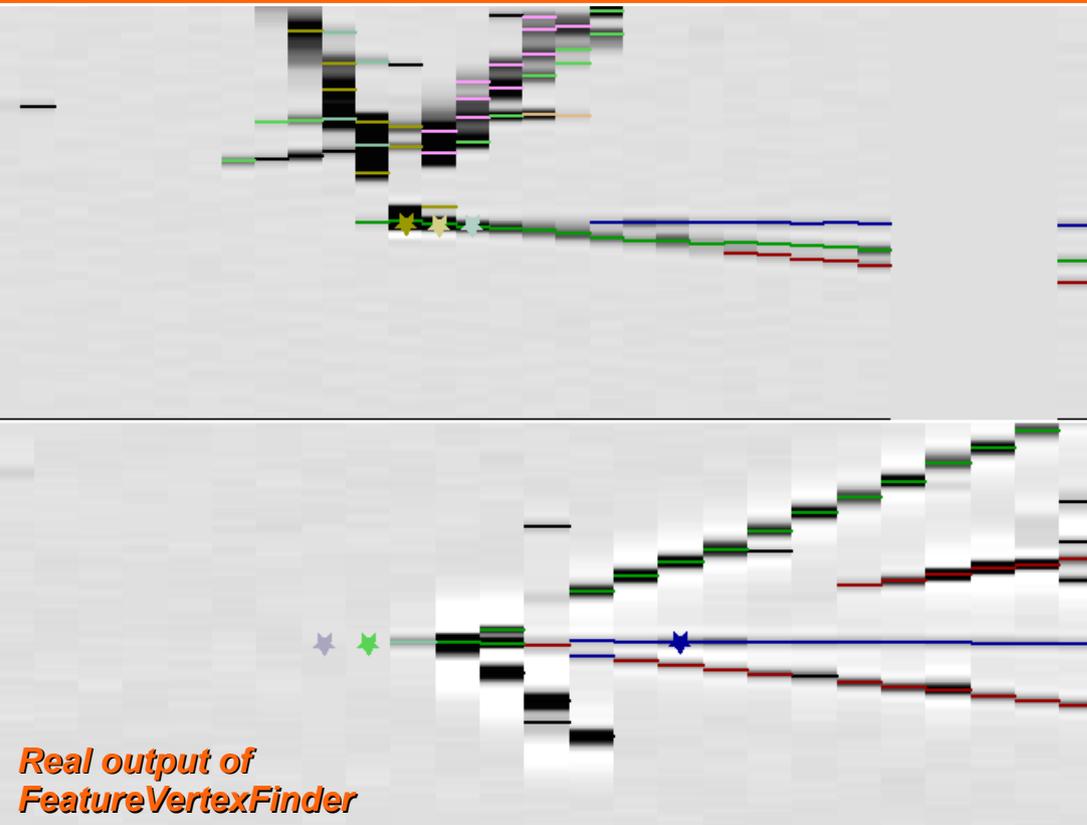
Feature Vertex Finder

Vertex Strength

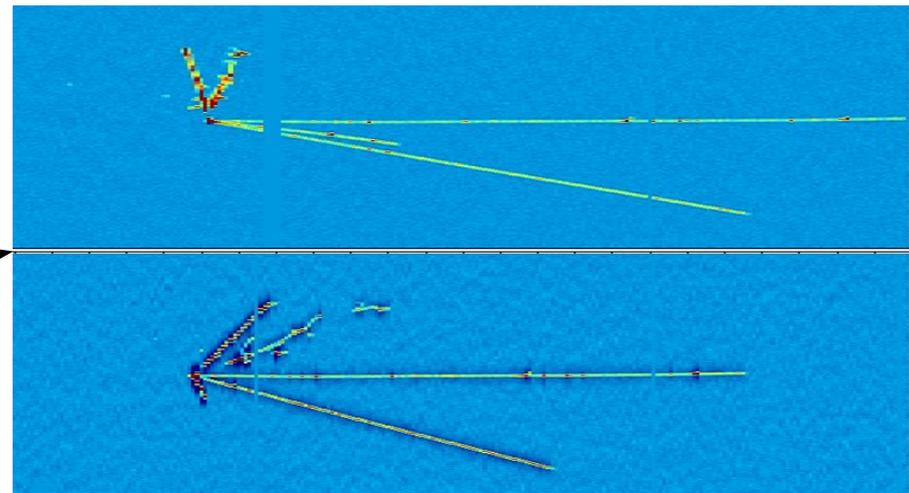
Recall that I have done this process for both startpoints and endpoints for every cluster

This means I have to do some bookkeeping to make sure I remove duplicate entries (*details in the backups*)

Now I need to assess how strong of a candidate each found 3d/2d vertex found



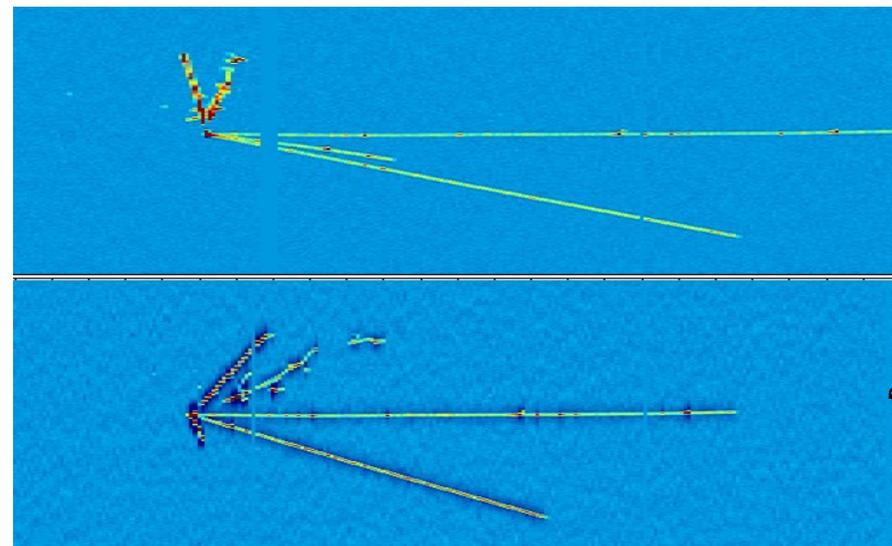
To do this I come back to the image techniques employed in CornerFinderAlg



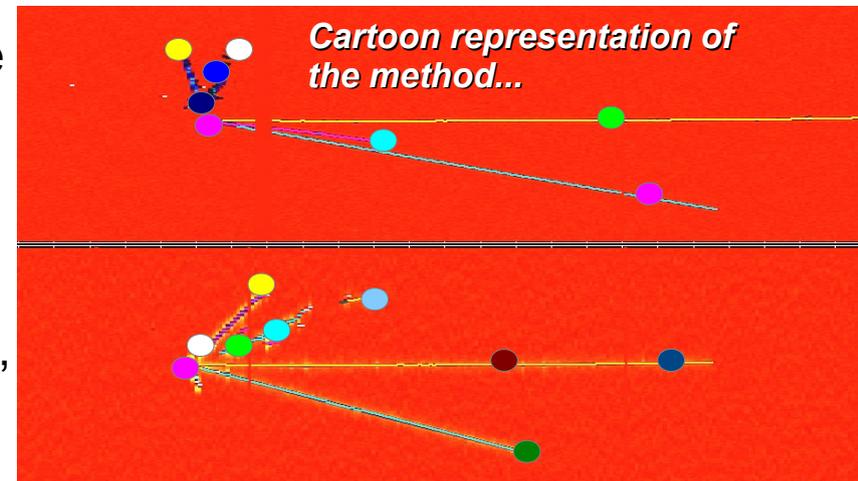
Feature Vertex Finder

CornerFinder

Lots more details about the CornerFinder have been given by W. Ketchum and can be found <https://cdcv.s.fnal.gov/redmine/attachments/download/9953/CornerFinderIntro.pdf> & <https://indico.fnal.gov/getFile.py/access?contribId=1&resId=0&materialId=slides&confId=6845> and another example of it being used by B. Jones for tracks can be found <https://indico.fnal.gov/getFile.py/access?contribId=3&resId=0&materialId=slides&confId=6845>



Using the image techniques described elsewhere (see *back-up slides*) I find the 2-d “corner features” in each plane



This technique finds **lots** of 2d corner points (wire & time) in each plane

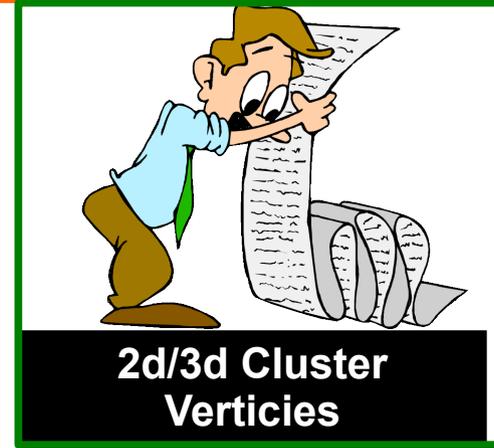
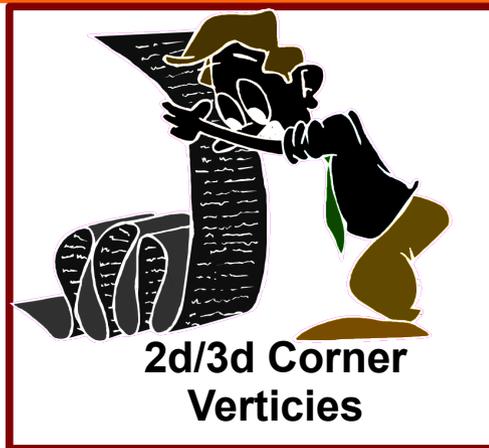
Now I loop over all the corner points in each plane and similar to before keep only the corner points that match in Wire and Time

(note: I also record the 3d matched corner points strength as defined by the CornerFinderAlg...this is important for later)

This gives me a list of 2d/3d corner vertices (x, y, z)

Feature Vertex Finder

A tale of two lists



Now what I have is two lists of 2d/3d vertex candidates (corners and clusters) found from two different methods

What comes next depends on the length of the cluster vertex list and the proximity in 3d space (x , y , z in cm) to the vertices from the corner vertex lists

Nitty gritty details can be found in the back-up slides, but everything follows this rough prescription

Feature Vertex Finder

A tale of two lists



2d/3d Cluster Vertices

- 0) All vertex candidates start with a strength equal to zero
- 1) Loop over the cluster vertex list and merge a 3d vertex that is within 0.5 cm (**new*) of another vertex in x, y, and z (has to satisfy all three spatial directions)
 - When you merge two vertex candidates +1 to the vertex strength
 - Merging right now is the dumb $(m+n)/2$...needs to be improved



2d/3d Corner Vertices

- 2) Take the merged cluster vertex list and compare it to the corner vertex list.
 - If there is a corner vertex within 1 cm (**new*) in x, y, z of the cluster vertex add +1 to the strength
 - I do not merge these vertex candidates...I just use them to add weight to the found cluster vertex

Note: Bug fix since last week now has the strength reporting "reasonable" numbers

- 3a) If you have > 0 vertex candidates, record all of them
 - EndPoint2d (TimeTick, geo::WireID, strength, Vtx #, View, Total Charge *not used*)
 - Vertex (xyz, Vtx #)



- 3b) If you have exactly 0 vertex candidates use a bail/recover method
 - More on the next slide
 - Nitty gritty details can be found in the back-up slides, but everything follows this rough prescription*

Feature Vertex Finder

Bailout



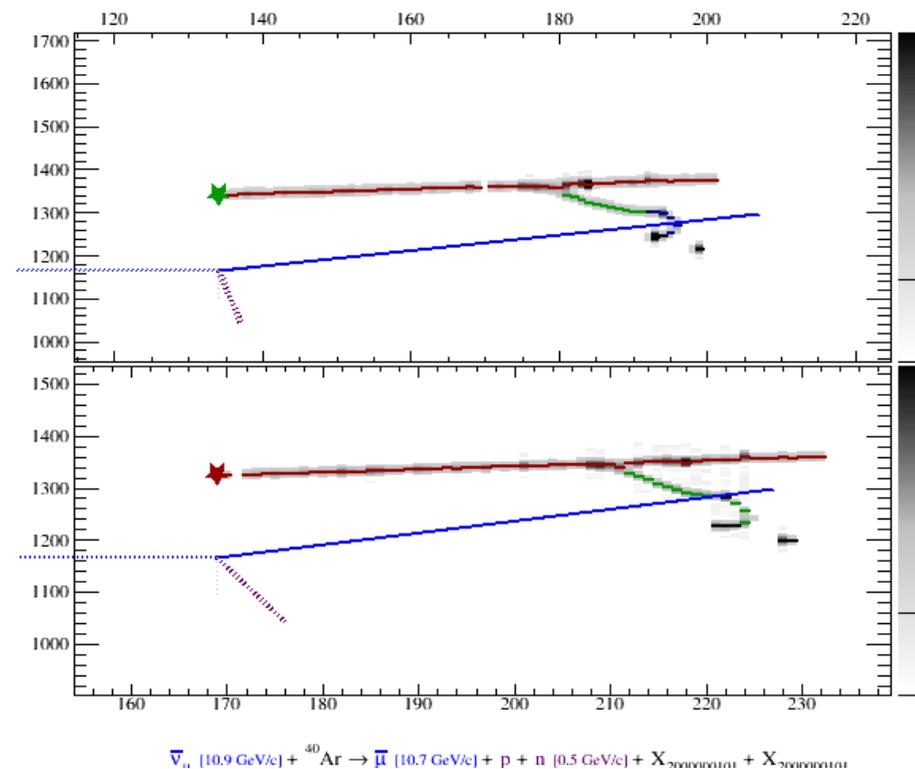
So if somehow after looking for a cluster vertex and/or a corner vertex we still haven't found at least one 3d proto-vertex we employ a series of bail out tactics

Note: All these verticies will have strength 0 or 1

Bail Out Strategy 1:

Take the start point and end point of the longest cluster in each plane and try to match this 2d point to a corresponding 3d corner point projected down into the plane (if it matches to many 3d feature points use the strongest 3d feature point).

→ If you find a match take the strongest point and construct a 2d/3d proto-vertex from the 3d feature point (this way I can have a point in all planes that is consistent) **strength 1**



$\bar{\nu}_\mu [10.9 \text{ GeV/c}] + {}^{40}\text{Ar} \rightarrow \bar{\mu} [10.7 \text{ GeV/c}] + p + n [0.5 \text{ GeV/c}] + X_{\text{neutrinos}} + X_{\text{hadrons}}$

Feature Vertex Finder

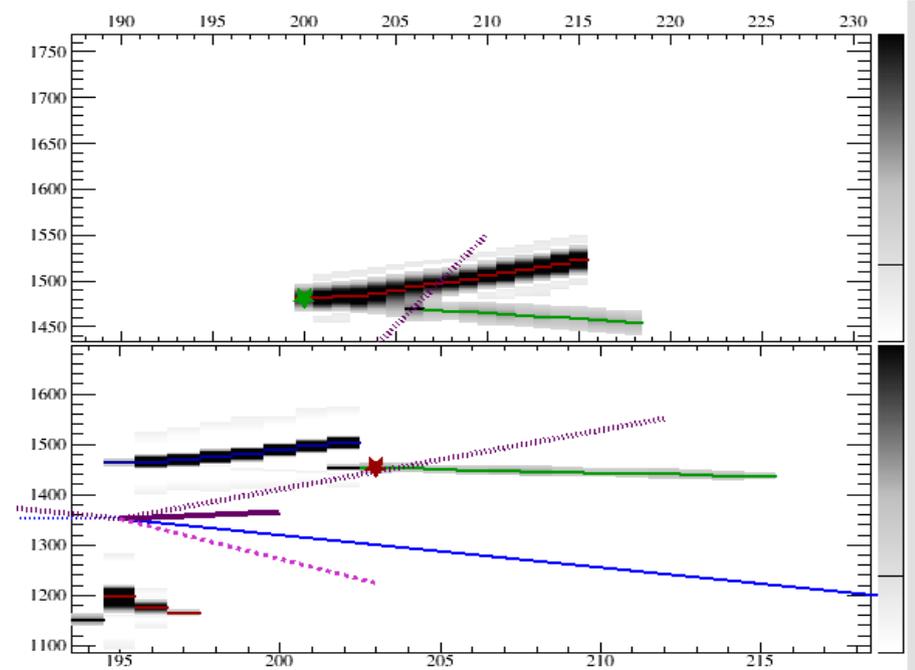
Bailout



Bail Out Strategy 2 (only used if strategy 1 fails):

Take the strongest 3d feature point found and project it down into 2d

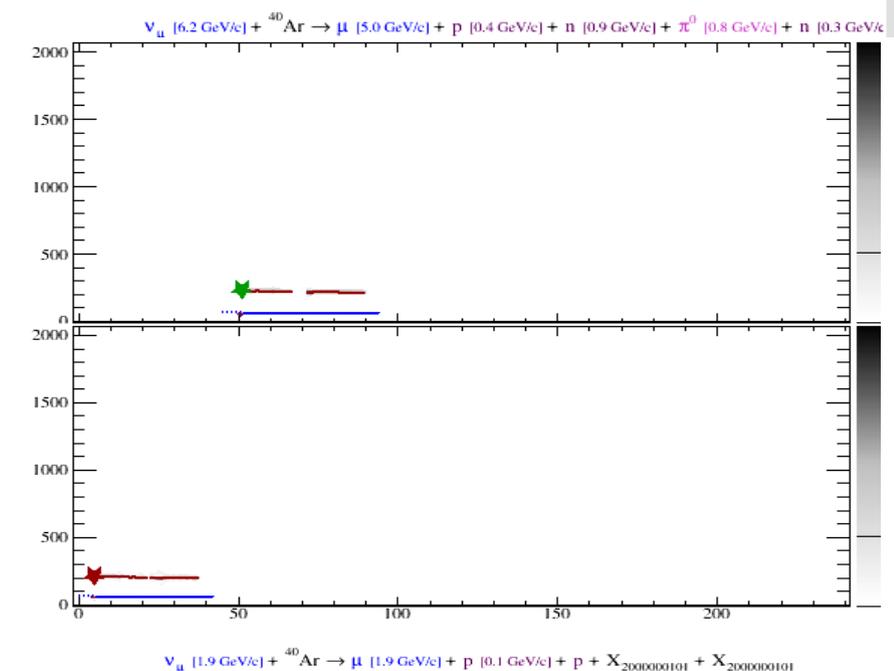
→ Take this point and construct a 2d/3d proto-vertex **strength 0**



Bail Out Strategy 3 (only used if strategy 1 & 2 fails):

Take the start point of the longest cluster in each plane as the 2d vertex (regardless of if they match between views)

→ Use geometry to find the nearest 3d point between the planes **strength 0**



Preliminary Performance Plots

How well does FeatureVertexFinder do?

→ **Look at 400 Genie events generated in ArgoNeuT**

- Reconstructed: GausHitFinder → dBCluster → HoughLineFinder → **LineMerger**

→ **First look at all reconstructed vertex candidates created**

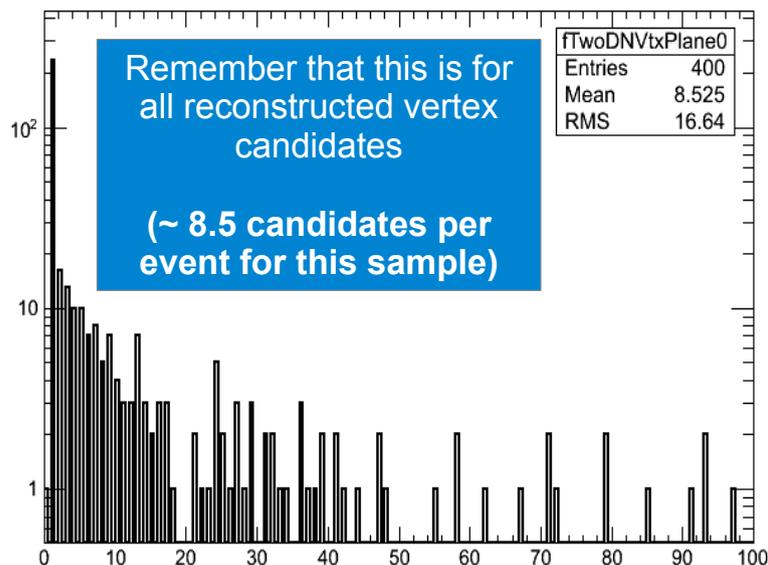
- Some good stuff...lots of interesting noise

→ **Then look to see what happens as we increase the strength of the 2d vertex reconstructed**

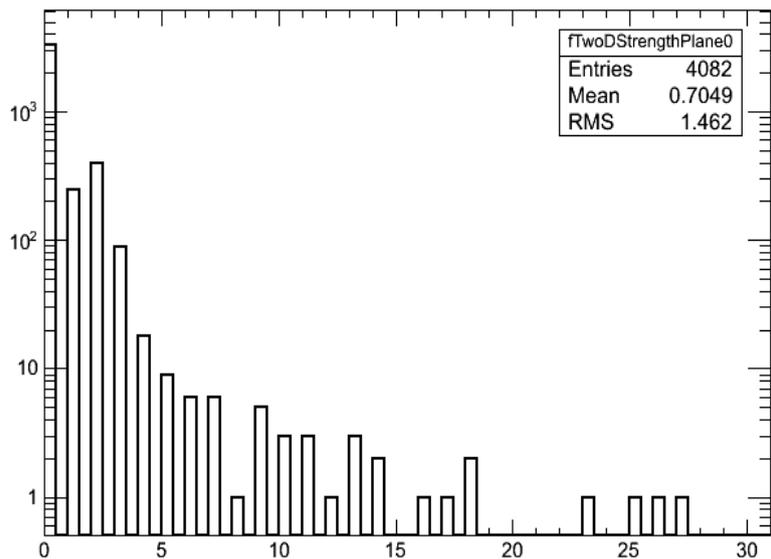
Preliminary Performance Plots

All reconstructed vertices

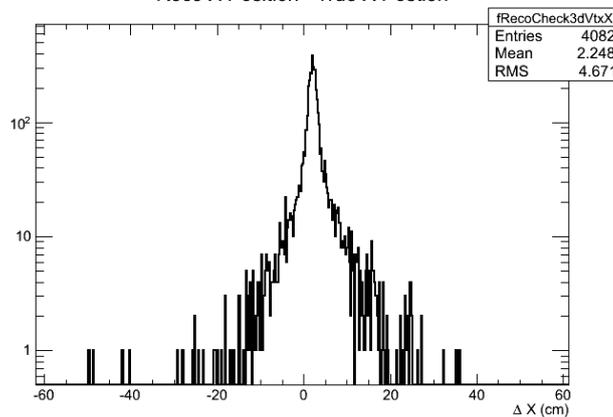
TwoD Number of Vertices Found in Plane 0



TwoD Strength Plane 0

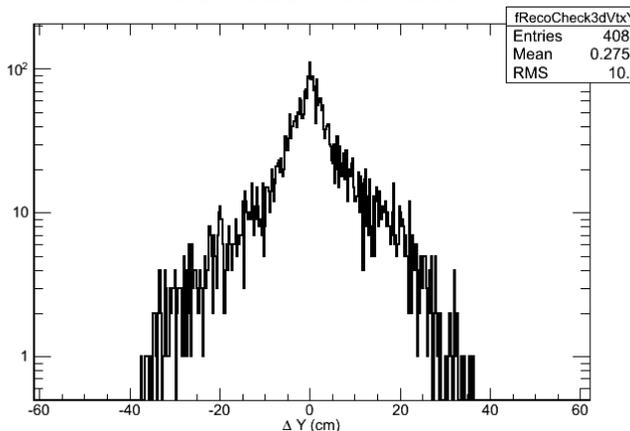


Reco X Position - True X Position



X offset because I need to handle ArgoNeuT's trigger offset properly still

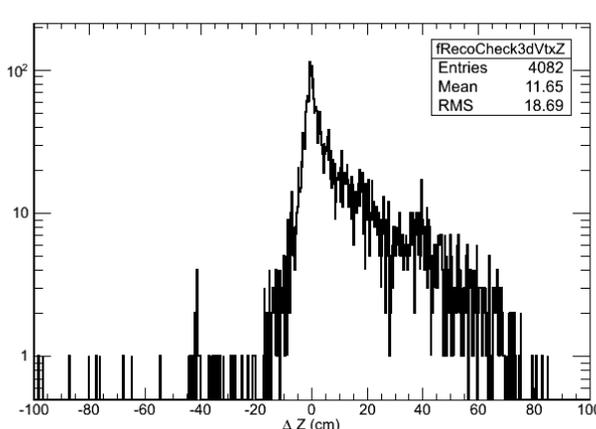
Reco Y Position - True Y Position



For this sample of 400 events (low statistics):

I reconstruct a vertex within 1 cm in wire distance and time tick (converted to cm) of the true vertex in both planes ~70% of the time

Reco Z Position - True Z Position

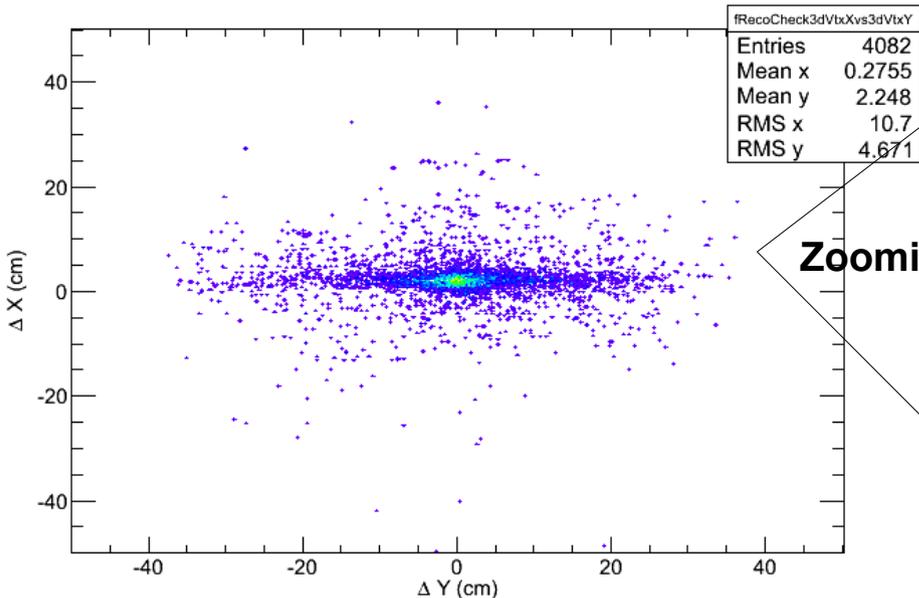


I reconstruct a 3d vertex within 1 cm in x, y, and z simultaneously of the true vertex only 60% of the time (my resolution in Y and Z seem to really lower my efficiency)

Preliminary Performance Plots

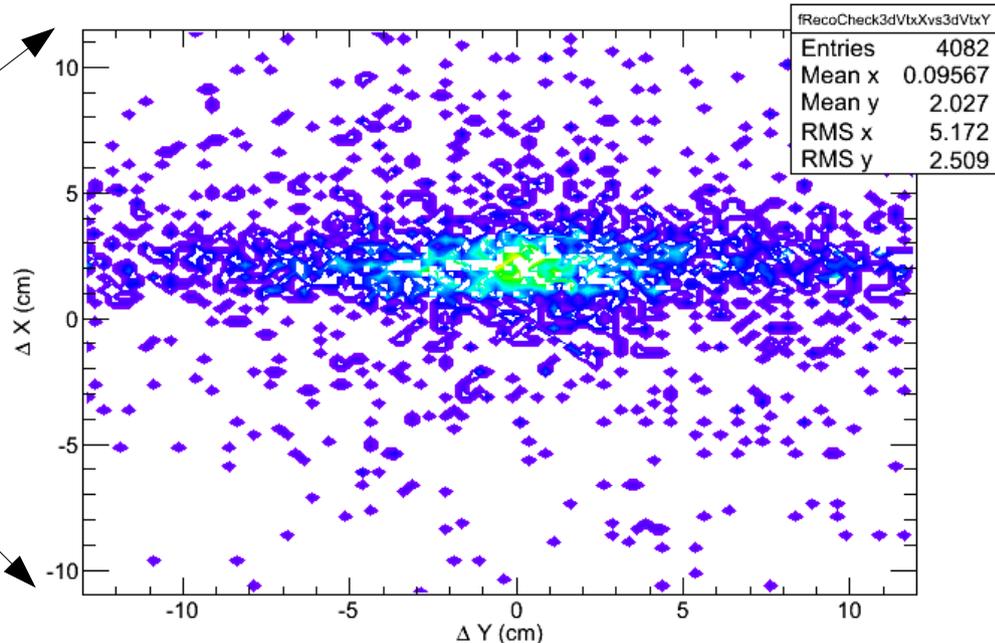
All reconstructed vertices

Delta X vs Delta Y

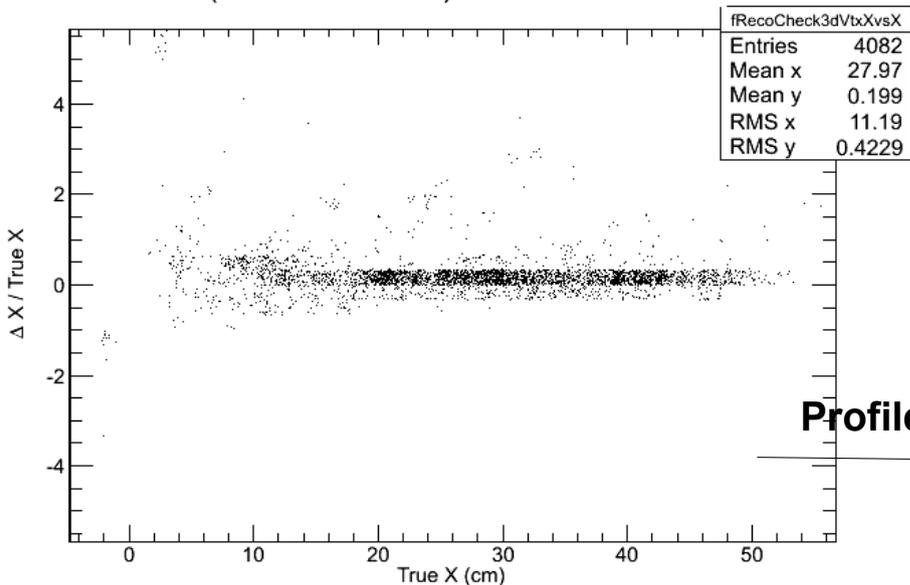


Zooming in

Delta X vs Delta Y

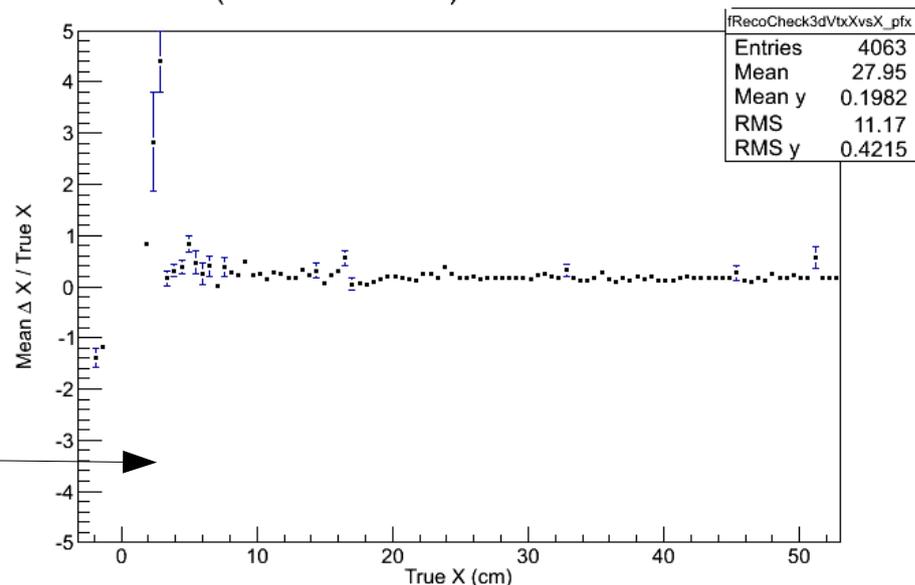


(Reco X - True X)/True X vs True X



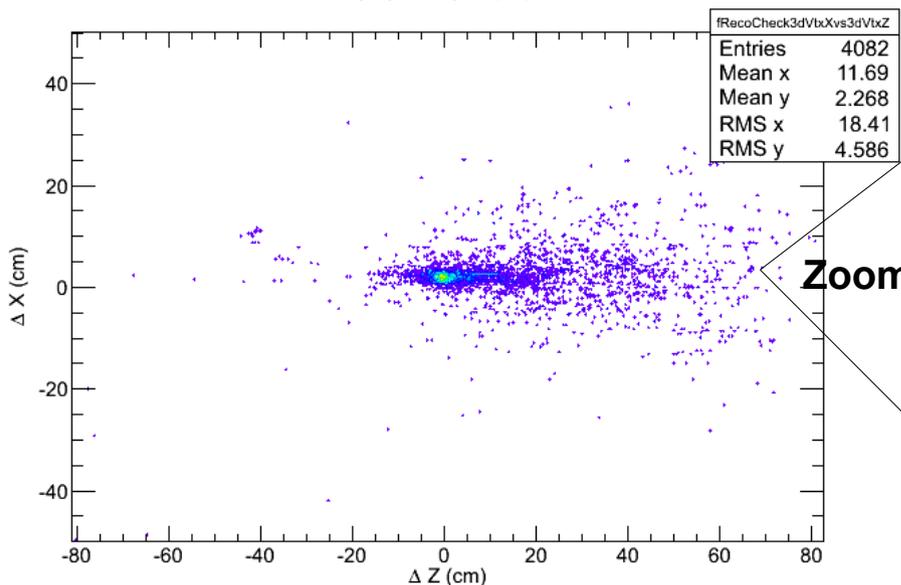
Profile

(Reco X - True X)/True X vs True X



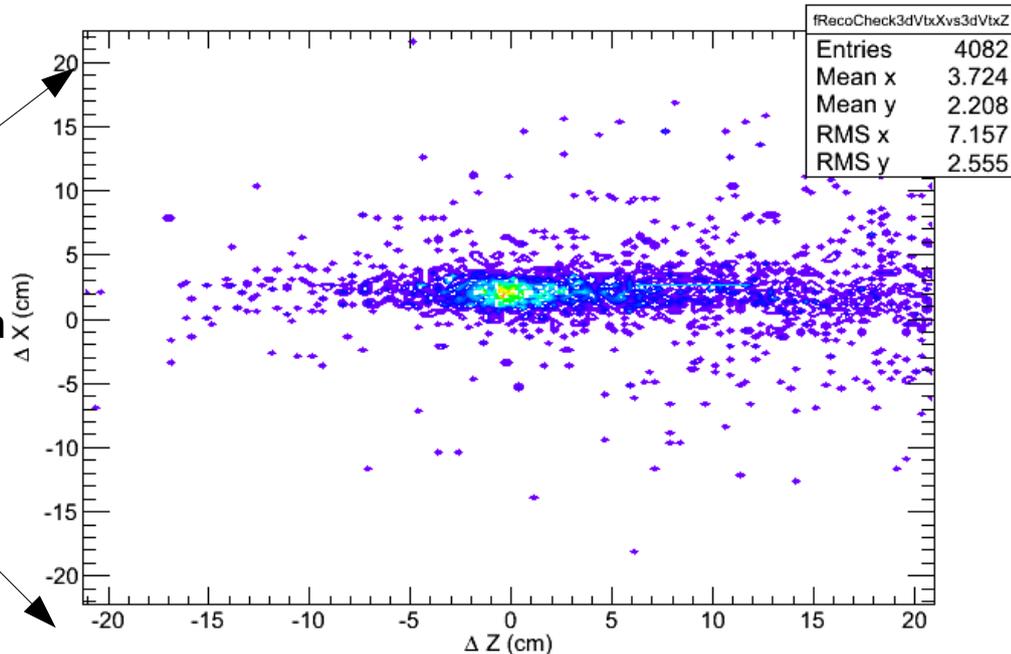
Preliminary Performance Plots

Delta X vs Delta Z

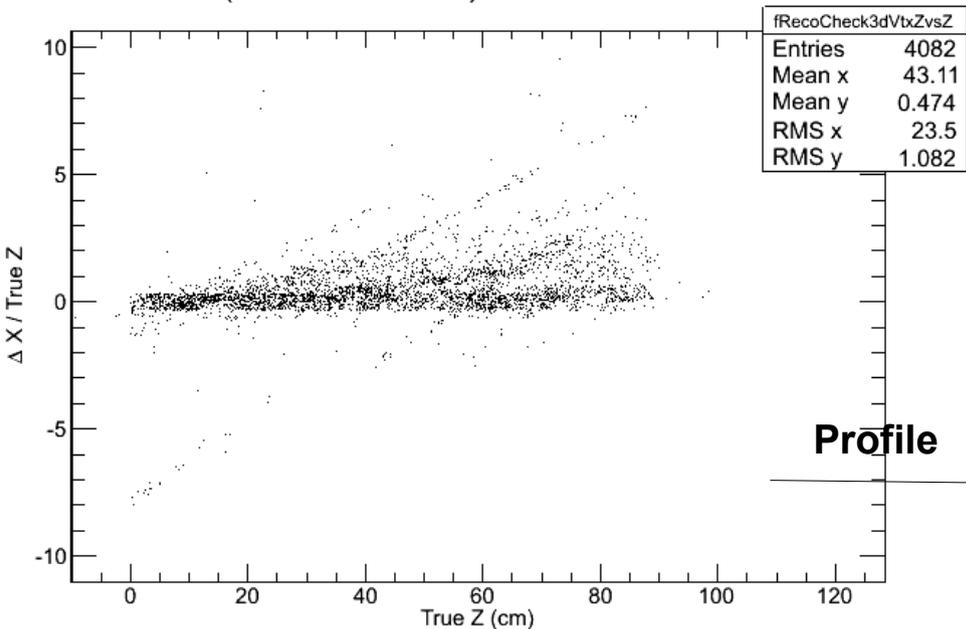


Zooming in

Delta X vs Delta Z

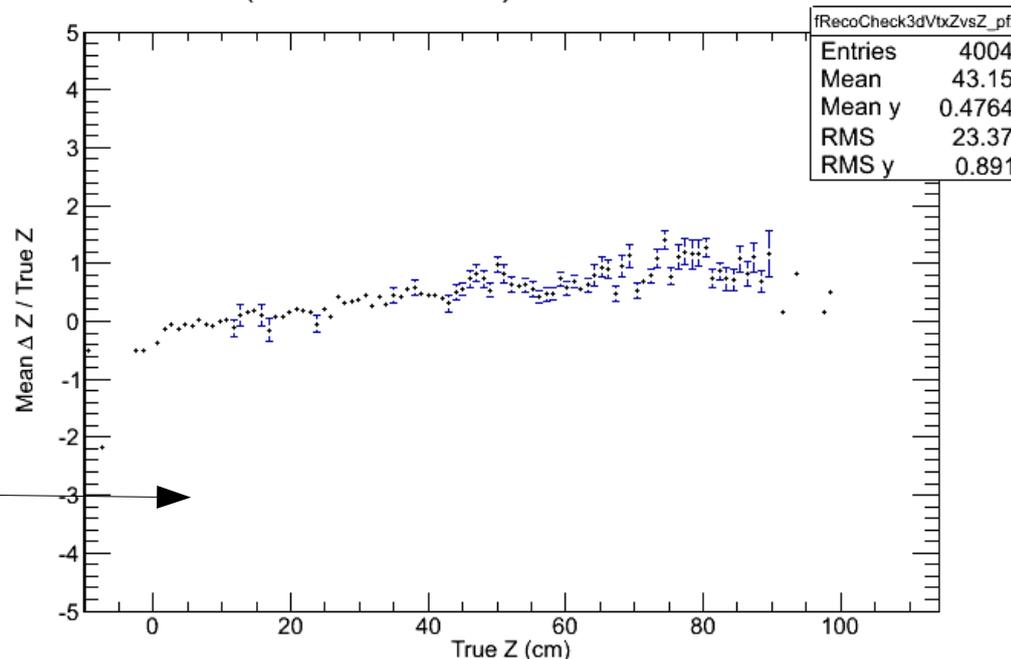


(Reco Z - True Z)/True Z vs True Z



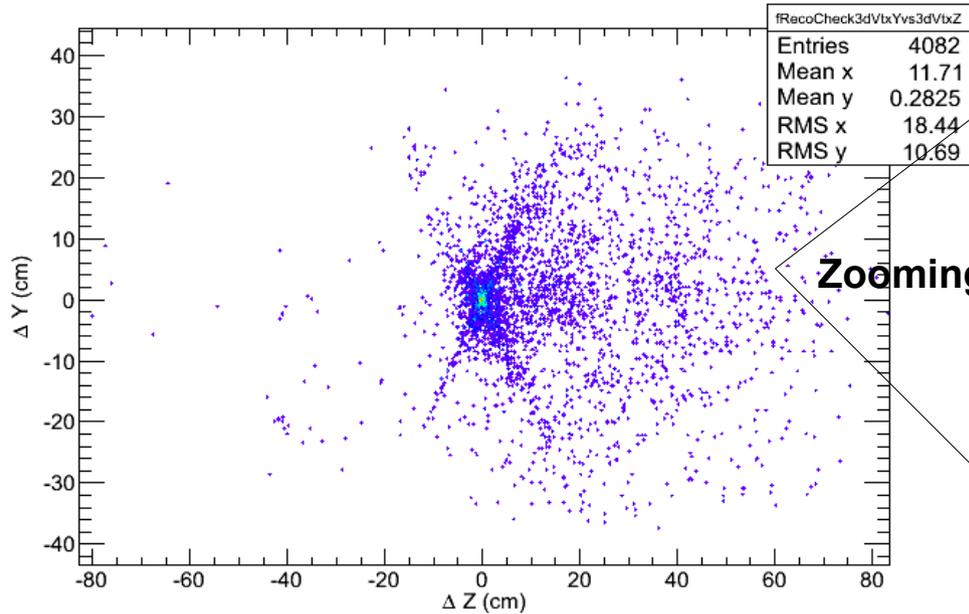
Profile

(Reco Z - True Z)/True Z vs True Z

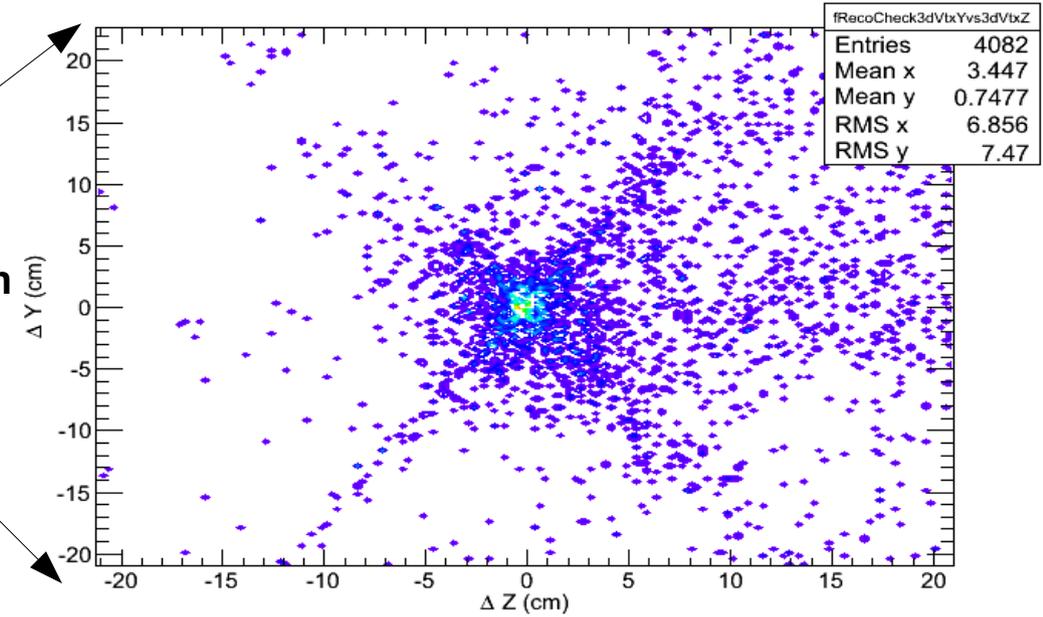


Preliminary Performance Plots

Delta Y vs Delta Z

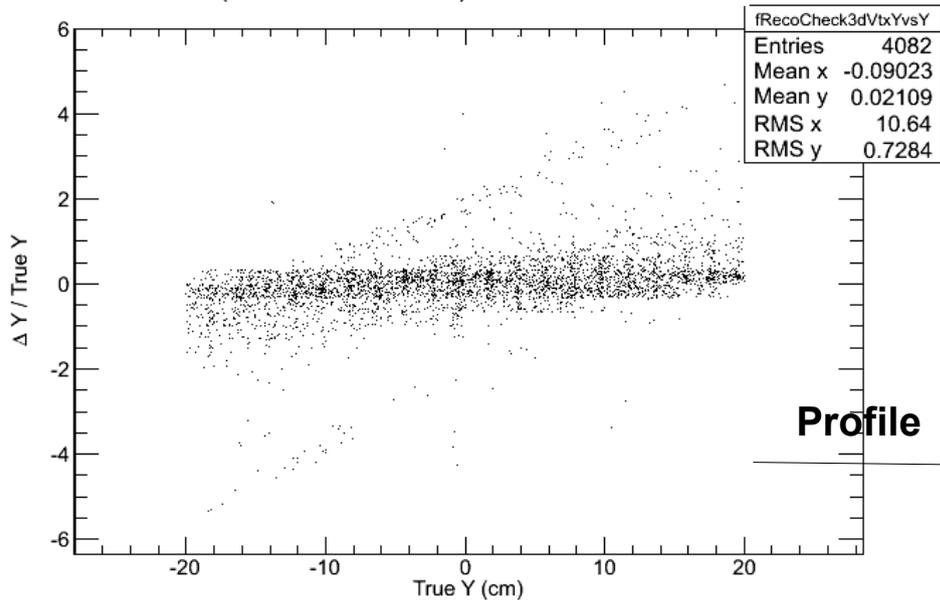


Delta Y vs Delta Z



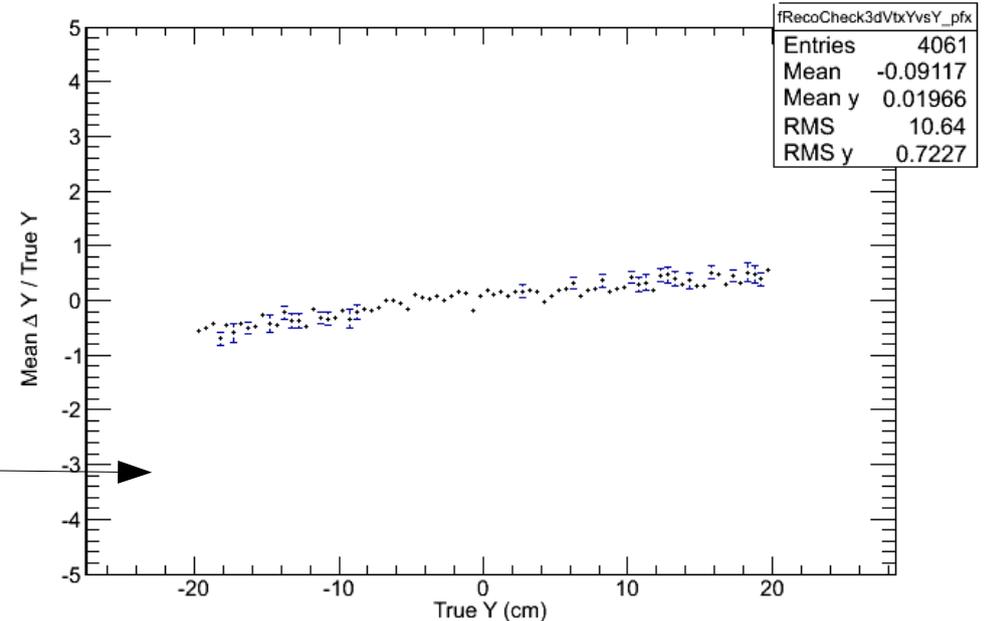
Zooming in

(Reco Y - True Y)/True Y vs True Y



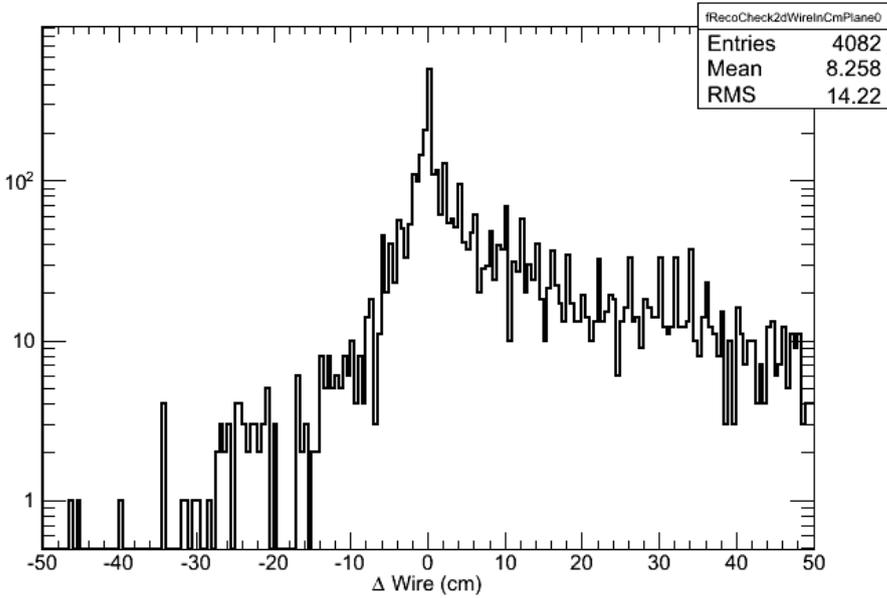
Profile

(Reco Y - True Y)/True Y vs True Y

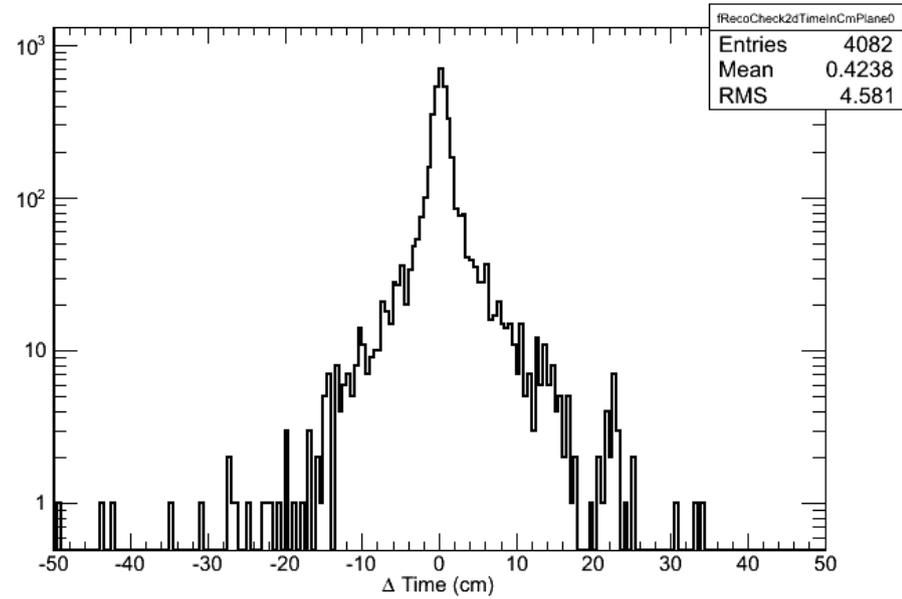


Preliminary Performance Plots

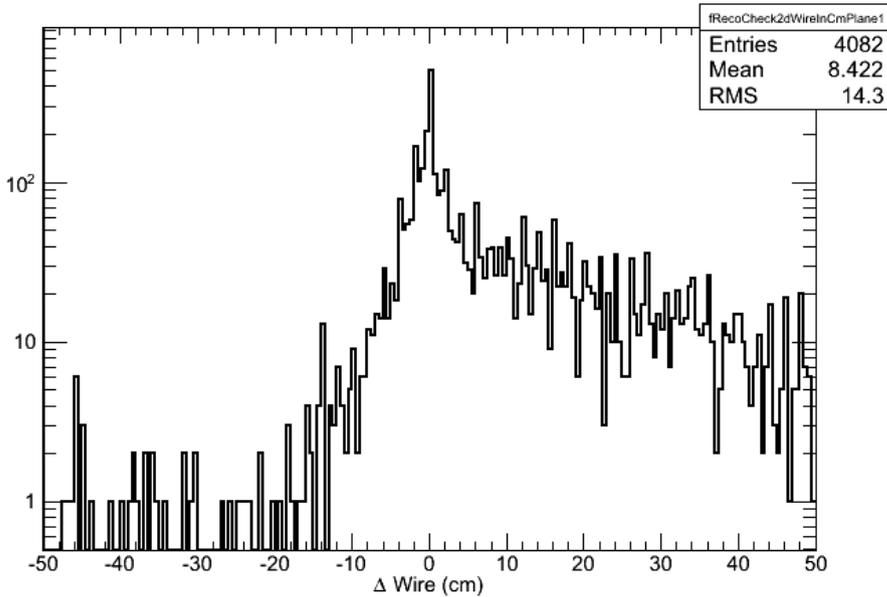
Reco Wire in CM - True Wire in CM Plane 0



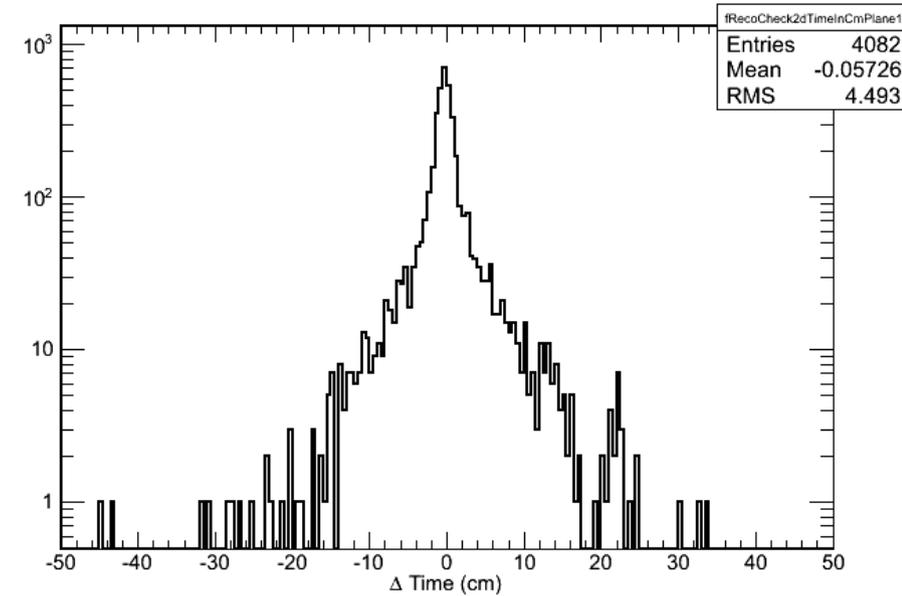
Reco Time in CM - True Time in CM Plane 0



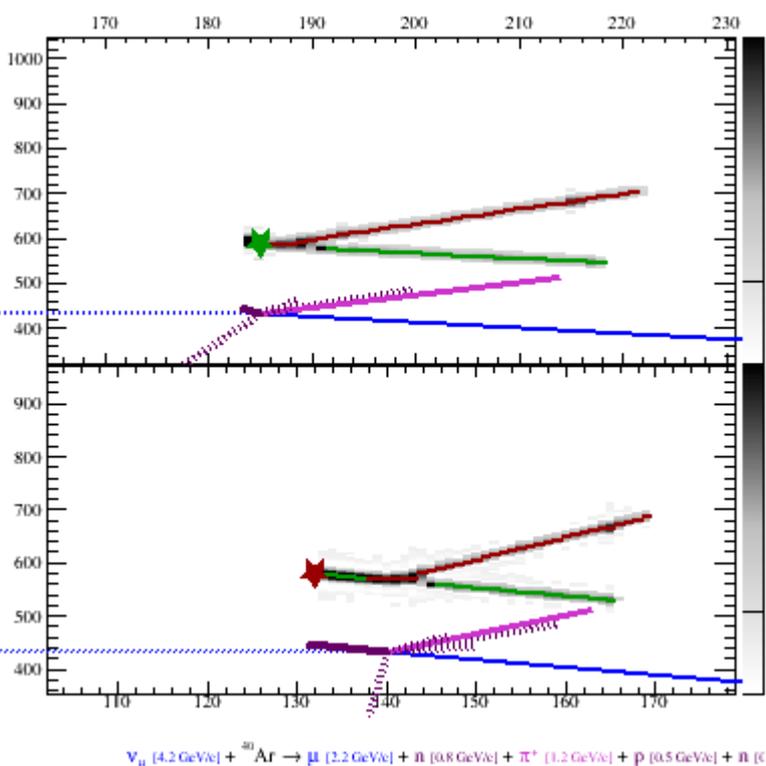
Reco Wire in CM - True Wire in CM Plane 1



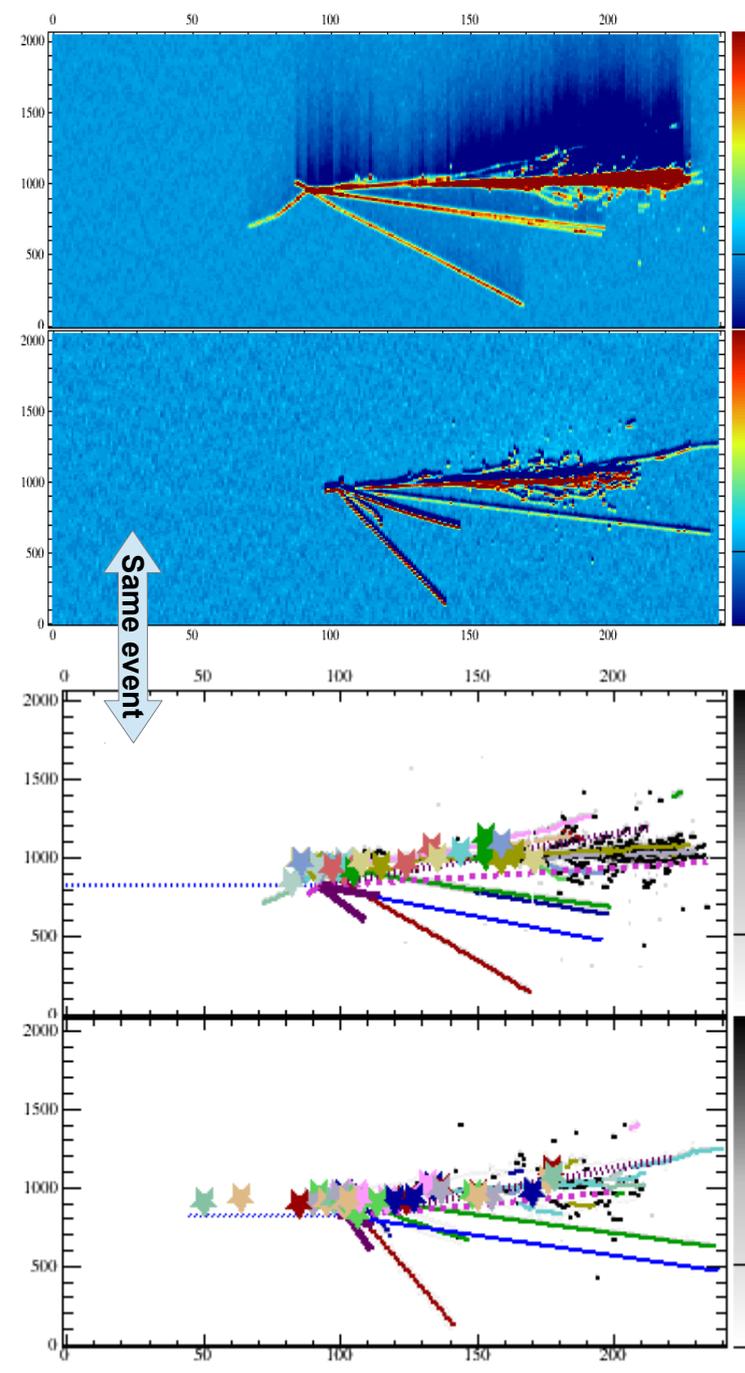
Reco Time in CM - True Time in CM Plane 1



Taste of everything we find...



It's true that we do find the primary vertex in nice simple events like this one



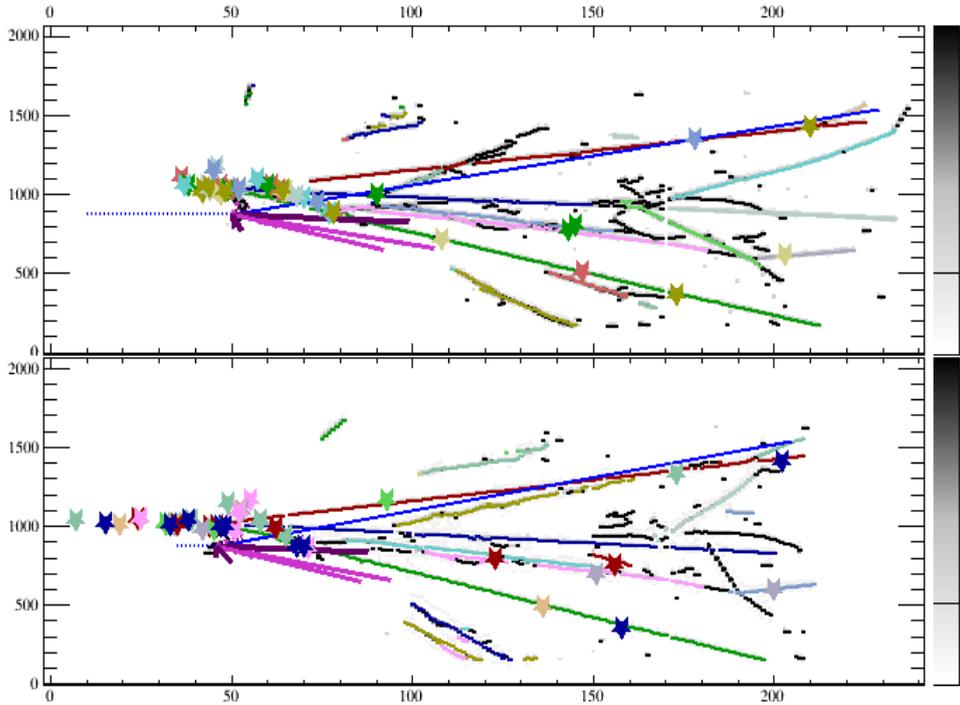
But there are also events like this where we find **lots** of proto-vertices

Don't want to throw away this information
 → Some of this is a function of what clustering algorithm you chose

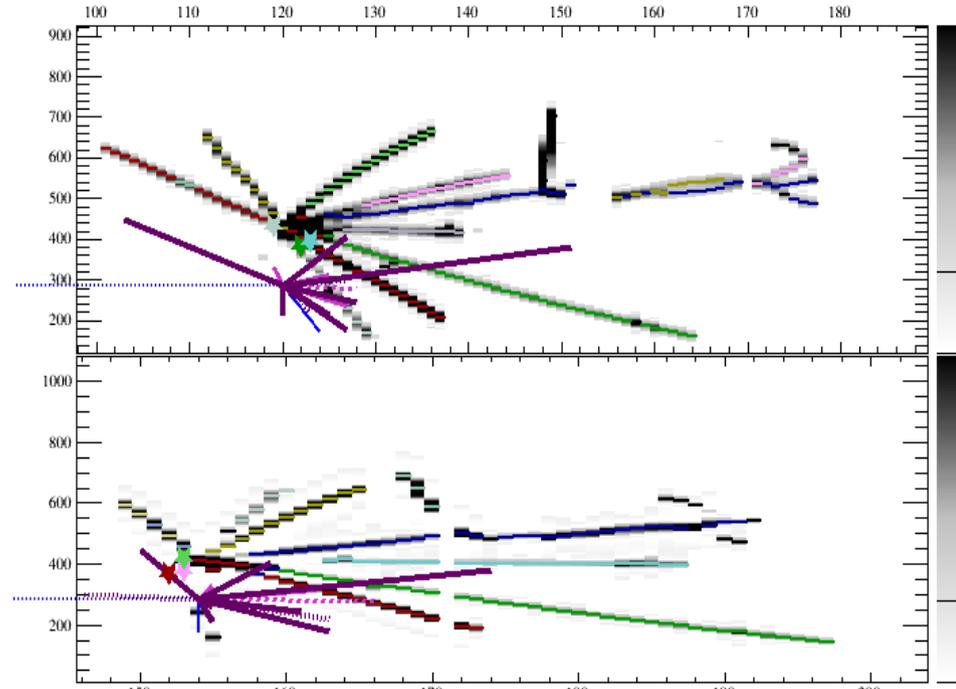
→ There is information in this that tells we likely have a shower in this event

ν_{μ} [23.9 GeV/c] + $^{40}\text{Ar} \rightarrow \mu$ [6.3 GeV/c] + π^0 [8.2 GeV/c] + n [8.0 GeV/c] + p [1.0 GeV/c] + π^+ [0.5 GeV/c]

Taste of everything we find...



V_{μ} [19.6 GeV/c] + $^{40}\text{Ar} \rightarrow \mu$ [12.2 GeV/c] + π^+ [0.1 GeV/c] + p [0.3 GeV/c] + π^- [2.5 GeV/c] + p [2.1 GeV/c] + π^0 [0.6 GeV/c] +



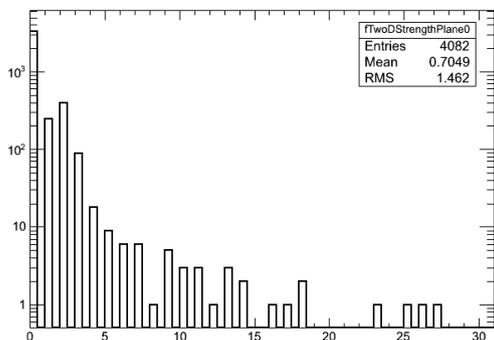
V_{μ} [3.5 GeV/c] + $^{40}\text{Ar} \rightarrow \mu$ [0.4 GeV/c] + π^- [0.2 GeV/c] + n [0.3 GeV/c] + π^+ [0.3 GeV/c] + π^0 [0.5 GeV/c] + p [1.3 GeV/c] +

Again, taking everything can be bad...

But can also be good...

EndPoint2d Strength

TwoD Strength Plane 0

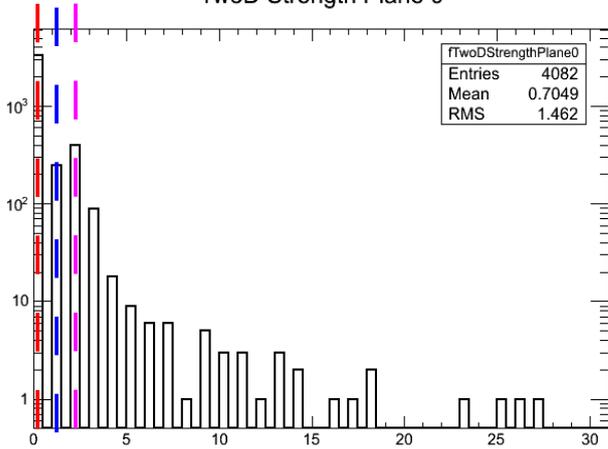


What I'd really like to be able to do is to make a plot of the 3d vertex "strength"

- Then I could cut / require the 3d vertex strength to be above a certain number
- I could take the strongest vertex as the "primary" vertex....or code it to be with some likelihood
- However the RecoBase Vertex does not have this member

More Performance Plots

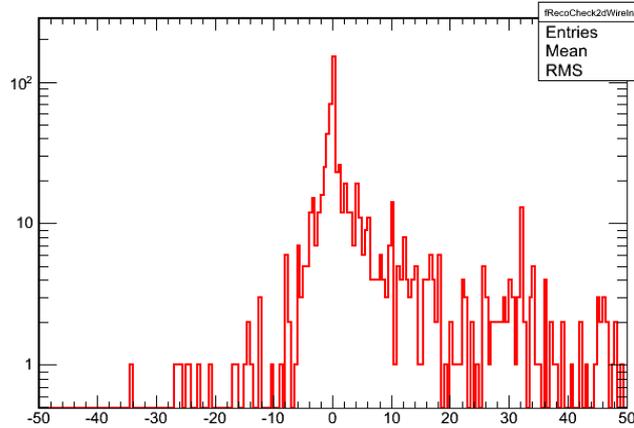
TwoD Strength Plane 0



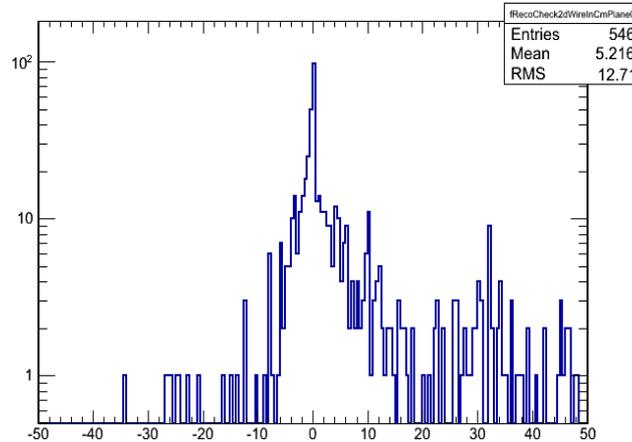
In lieu of modifying RecoBase/Vertex, what if I only plot the 2d results for EndPoint2d vertices (which all must correspond to a 3d vertex) above a certain strength (say, \geq 1, 2, 3)

Might be ok if I was going for purity...but not very good for efficiency

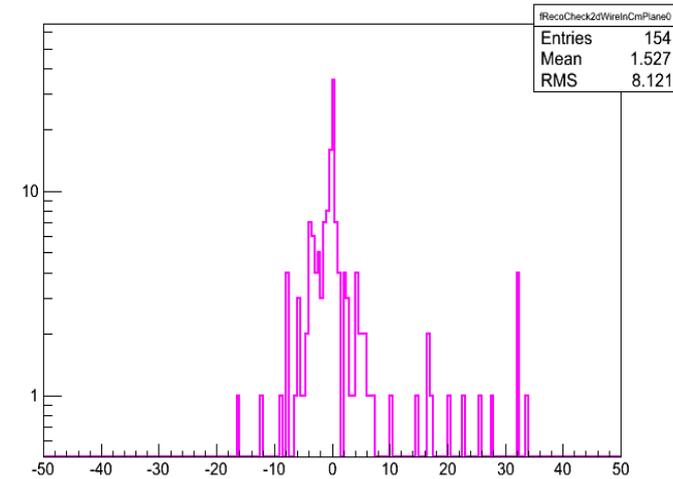
Reco Wire in CM - True Wire in CM Plane 0



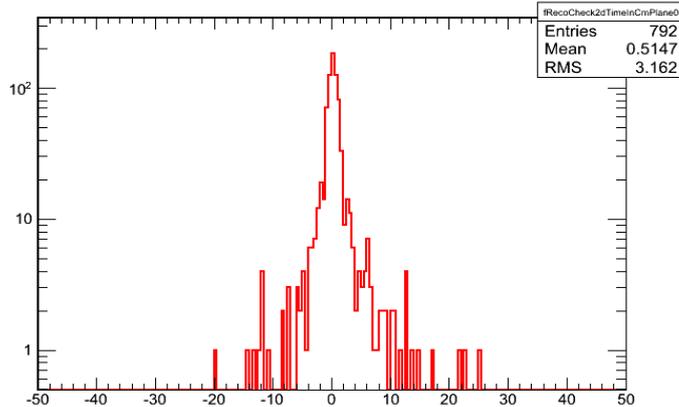
Reco Wire in CM - True Wire in CM Plane 0



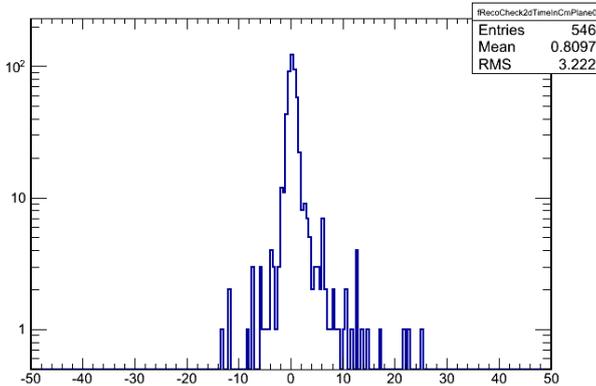
Reco Wire in CM - True Wire in CM Plane 0



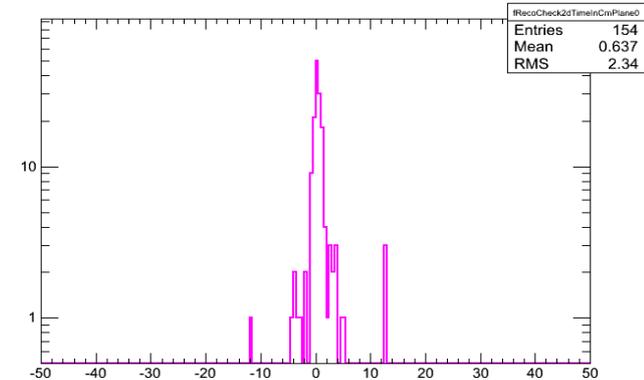
Reco Time in CM - True Time in CM Plane 0



Reco Time in CM - True Time in CM Plane 0



Reco Time in CM - True Time in CM Plane 0



Conclusions

- **FeatureVertexFinder** is in LArSoft right now along with **FeatureVertexFinderAna** (used to produce all the plots you see in this talk)
 - You should check it out and give it a whirl
 - Let me know problems and improvements you think are needed (there are lots...)
- **Planned improvements / studies**
 - Producing plots in MicroBooNE
 - Lower energy neutrino beam might have things look better
 - Utilizing 3 plane geometry to hopefully find the primary vertex more reliably
 - Study the difference given various clustering algorithms
 - HoughLines, FuzzyCluster, dBCluster, etc...
 - Work on tuning merging / matching parameters
 - Might have to depend on the topology of the event...
 - Want to incorporate proximity to hits as a way of increasing the vertex strength
 - Need to be smart about this so as not to miss neutral current events
 - Open to collaboration with others on the code / suggestions to making our reconstruction work more dynamically
- **Formal request to change RecoBase/Vertex members**

Back-up Slides

Feature Vertex Finder

2-d Slopes

BELOW IS THE CODE FOR CALCULATING INTERCEPTS FROM SLOPES

```
// #####
// ### Now we try to find a 2-d vertex in the plane we are currently looking in ###
// #####
for (unsigned int n = nClustersFound; n > 0; n--)
{
  // #####
  // ### Looping over the clusters starting from the ###
  // ### first cluster and checking against the nCluster ###
  // #####
  for (unsigned int m = 0; m < n; m++)
  {
    // #####
    // ### Checking to make sure clusters are in the same view ###
    // #####
    if(Clu_Plane[n] == Clu_Plane[m])
    {
      // --- Skip the vertex if the lines slope don't intercept ---
      if(Clu_Slope[m] - Clu_Slope[n] == 0){break;}
      // =====
      // === X intersection = (yInt2 - yInt1) / (slope1 - slope2) ===
      float intersection_X = (Clu_Yintercept[n] - Clu_Yintercept[m]) / (Clu_Slope[m] - Clu_Slope[n]);
      // =====
      // === Y intersection = (slope1 * XInt) + yInt1 ===
      float intersection_Y = (Clu_Slope[m] * intersection_X) + Clu_Yintercept[m];
      // ### Filling the vector of Vertex Wire, Time, and Plane ###
      // -----
      // --- Skip this vertex if the X and Y intersection is outside the detector ---
      // --- using geom->Nwires(plane,tpc,cyostat) & detprop->NumberTimeSamples() ---
      // -----
      if( intersection_X > 1 && intersection_Y > 0 &&
          (intersection_X < geom->Nwires(Clu_Plane[n],0,0) || intersection_X < geom->Nwires(Clu_Plane[m],tpc,cstat) ) &&
          intersection_Y < detprop->NumberTimeSamples() )
      {
        vtx_wire.push_back(intersection_X);
        vtx_time.push_back(intersection_Y);
        vtx_plane.push_back(Clu_Plane[m]);
        n2dVertexCandidates++;
      }
      //<!--End saving a "good 2d vertex" candidate
      // =====
      // === X intersection = (yInt2 - yInt1) / (slope1 - slope2) ===
      float intersection_X2 = (Clu_Yintercept2[n] - Clu_Yintercept2[m]) / (Clu_Slope[m] - Clu_Slope[n]);
      // =====
      // === Y intersection = (slope1 * XInt) + yInt1 ===
      float intersection_Y2 = (Clu_Slope[m] * intersection_X) + Clu_Yintercept2[m];
      // #####
      // ### Filling the vector of Vertex Wire, Time, and Plane ###
      // -----
      // --- Skip this vertex if the X and Y intersection is outside the detector ---
      // --- using geom->Nwires(plane,tpc,cyostat) & detprop->NumberTimeSamples() ---
      // -----
      if( intersection_X2 > 1 && intersection_Y2 > 0 &&
          (intersection_X2 < geom->Nwires(Clu_Plane[n],0,0) || intersection_X2 < geom->Nwires(Clu_Plane[m],tpc,cstat) ) &&
          intersection_Y2 < detprop->NumberTimeSamples() )
      {
        vtx_wire.push_back(intersection_X2);
        vtx_time.push_back(intersection_Y2);
        vtx_plane.push_back(Clu_Plane[m]);
        n2dVertexCandidates++;
      }
      //<!--End saving a "good 2d vertex" candidate
      //<!--End making sure we are in the same plane
    }
  }
}
//<!-- End n-- loop
```

Checking to make sure the vertices are in the same plane

Calculate the x (wire) and y (time) of the intersection point for the start point of the cluster and the end point of the cluster

then just make sure the vertex makes sense (inside the detector)

Feature Vertex Finder

Removing Duplicates

```
// #####  
// ### Now we need to make sure that we remove duplicate vertices ###  
// ### just in case we have any in our list of n3dVertex ###  
// #####  
double x_3dVertex_dupRemoved[100000] = {0.}, y_3dVertex_dupRemoved[100000] = {0.}, z_3dVertex_dupRemoved[100000] = {0.};  
int n3dVertex_dupRemoved = 0;  
  
for(size_t dup = 0; dup < n3dVertex; dup ++)  
{  
    float tempX_dup = x_3dVertex[dup];  
    float tempY_dup = y_3dVertex[dup];  
    float tempZ_dup = z_3dVertex[dup];  
  
    bool duplicate_found = false;  
  
    for(size_t check = dup+1; check < n3dVertex; check++)  
    {  
        // #####  
        // ### I am going to call a duplicate vertex one that matches in x, y, and z ###  
        // ### within 0.1 cm for all 3 coordinates simultaneously ###  
        // #####  
        if(std::abs( x_3dVertex[check] - tempX_dup ) < 0.01 && std::abs( y_3dVertex[check] - tempY_dup ) < 0.01 &&  
            std::abs( z_3dVertex[check] - tempZ_dup ) < 0.01 )  
        {  
            duplicate_found = true;  
  
            }/!----End checking to see if this is a duplicate vertex  
  
        }/!----End check for loop  
  
        // #####  
        // ### If we didn't find a duplicate then lets save this 3d vertex as ###  
        // ### a real candidate for consideration ###  
        // #####  
        if(!duplicate_found)  
        {  
            x_3dVertex_dupRemoved[n3dVertex_dupRemoved] = tempX_dup;  
            y_3dVertex_dupRemoved[n3dVertex_dupRemoved] = tempY_dup;  
            z_3dVertex_dupRemoved[n3dVertex_dupRemoved] = tempZ_dup;  
  
            n3dVertex_dupRemoved++;  
        }  
  
    }/!----End dup for loop
```

Note:
Right now I consider a duplicate vertex any two 3d vertices that are within 0.01 cm in x, y, and z simultaneously

Feature Vertex Finder

A tale of two lists

```
double TwoDvertexStrength = 0;
// ### Case 1...only one 3d vertex found ###
if (n3dVertex_dupRemoved == 1)
{
// #####
// ### Looping over cryostats ###
// #####
for(size_t cstat = 0; cstat < geom->Ncryostats(); ++cstat)
{
// #####
// ### Looping over TPC's ###
// #####
for(size_t tpc = 0; tpc < geom->Cryostat(cstat).NTPC(); ++tpc)
{
// #####
// ### Loop over the wire planes ###
// #####
for (size_t i = 0; i < geom->Cryostat(cstat).TPC(tpc).Nplanes(); ++i)
{
double xyz[3] = {x_3dVertex_dupRemoved[0], y_3dVertex_dupRemoved[0], z_3dVertex_dupRemoved[0]};
// #####
// ### Give the current 3d vertex found a strength of 1 ###
// ### We will add +1 to it for each 3d feature that is ###
// ### near by it in x, y, z space ###
// #####
TwoDvertexStrength = 1;

// #####
// ### Does this point correspond to a 3d-Feature Point? ###
// ### Loop over all the feature points ###
// #####
for (int a = 0 ; a < n3dFeatures; a++)
{
// #####
// ### If you find a feature point within 3 cm of the vertex ###
// ### add a point to the vertex strength ###
// #####
if ( std::abs(xyz[0] - x_feature[a]) < 3 &&
std::abs(xyz[1] - y_feature[a]) < 3 &&
std::abs(xyz[2] - z_feature[a]) < 3 )
{
TwoDvertexStrength++;
}
}
}
}

double EndPoint2d_TimeTick = detprop->ConvertXTToTicks(xyz[0],i, tpc, cstat);

int EndPoint2d_Wire = geom->NearestWire(xyz , i, tpc, cstat);
int EndPoint2d_Channel = geom->NearestChannel(xyz, i, tpc, cstat);
geo::View_t View = geom->View(EndPoint2d_Channel);
geo::WireID wireID(cstat,tpc,i,EndPoint2d_Wire);
// ### Saving the 2d Vertex found ###
recob::EndPoint2D vertex( EndPoint2d_TimeTick , //<---TimeTick
wireID , //<---geo::WireID
TwoDvertexStrength , //<---Vtx strength (JA: ?)
epcol->size() , //<---Vtx ID (JA: ?)
View , //<---Vtx View
1 ); //<---Total Charge (JA: Need to figure this one?)

epcol->push_back(vertex);
} //<---End loop over Planes
} //<---End loop over tpc's
} //<---End loop over cryostats

// #####
// ### Saving the 3d vertex ###
// #####
double xyz2[3] = {x_3dVertex_dupRemoved[0], y_3dVertex_dupRemoved[0], z_3dVertex_dupRemoved[0]};
recob::Vertex the3Dvertex(xyz2, vcol->size());
vcol->push_back(the3Dvertex);

} //<---End Case 1, only one 3d Vertex found
```

This is the code I use when after searching for for cluster vertex candidates I have only found == 1 3d vertex

By default this only has strength = 1, but if it matches within 3cm of a corner vertex it gets +1 to the strength

I simply take this vertex, project down into 2-d and record the point

Feature Vertex Finder

A tale of two lists

```
if (n3dVertex_dupRemoved > 1)
{
    TwoDvertexStrength = 1;
    //std::cout<<" ### In case 2 ###"<<std::endl;
    // #####
    // ### Setting a limit to the number of merges ###
    // ### to be 3 times the number of 3d vertices found ###
    // #####
    int LimitMerge = 3 * n3dVertex_dupRemoved;
    // ### Trying to merge nearby vertices found ###
    for( int merge1 = 0; merge1 < n3dVertex_dupRemoved; merge1++)
    {
        for( int merge2 = n3dVertex ; merge2 > merge1; merge2--)
        {
            double temp1_x = x_3dVertex[merge1];
            double temp1_y = y_3dVertex[merge1];
            double temp1_z = z_3dVertex[merge1];

            double temp2_x = x_3dVertex[merge2];
            double temp2_y = y_3dVertex[merge2];
            double temp2_z = z_3dVertex[merge2];

            if( temp1_x == 0 || temp1_y == 0 || temp1_z == 0 ||
                temp2_x == 0 || temp2_y == 0 || temp2_z == 0) {continue;}

            // ### Merge the vertices if they are within 1.5 cm of each other ###
            if ( (std::abs( temp1_x - temp2_x ) < 1.0 && temp1_x != 0 && temp2_x != 0) &&
                (std::abs( temp1_y - temp2_y ) < 1.0 && temp1_y != 0 && temp2_y != 0) &&
                (std::abs( temp1_z - temp2_z ) < 1.0 && temp1_z != 0 && temp2_z != 0) &&
                nMerges < LimitMerge)
            {
                //std::cout<<" Yup, I am going to merge these! "<<std::endl;
                //std::cout<<" I've now performed a merge " <<nMerges<<" times"<<std::endl;
                //std::cout<<"Merging"<<std::endl;
                //std::cout<<"n3dVertex = " <<n3dVertex<<std::endl;
                nMerges++;
                // ### Zero the vertex that I am merging ###
                x_3dVertex[merge2] = 0.0;
                y_3dVertex[merge2] = 0.0;
                z_3dVertex[merge2] = 0.0;

                n3dVertex_dupRemoved++;

                // ### Add the merged vertex to the end of the vector ###
                x_3dVertex[n3dVertex_dupRemoved] = (temp1_x + temp2_x)/2;
                y_3dVertex[n3dVertex_dupRemoved] = (temp1_y + temp2_y)/2;
                z_3dVertex[n3dVertex_dupRemoved] = (temp1_z + temp2_z)/2;

                // #####
                // ### If we merged the vertices then increase its relative strength ###
                // #####
                TwoDvertexStrength++;
            }
        }
    }
}
//<---End merging vertices
//<---End merge2 loop
//<---End merge1 loop
```

Looping over the list of cluster vertices to look for things to merge

Merging if they are within 1.5 cm in x, y, and z

Zeroing the vertex you've just merged (these are removed from the list at a step not shown here)

Dumb $(x+y)/2$ merge