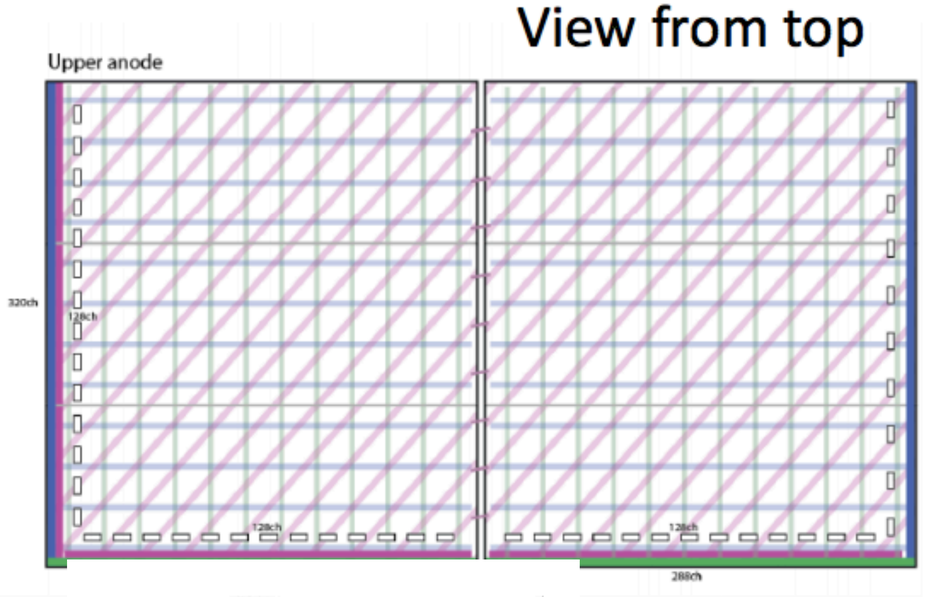
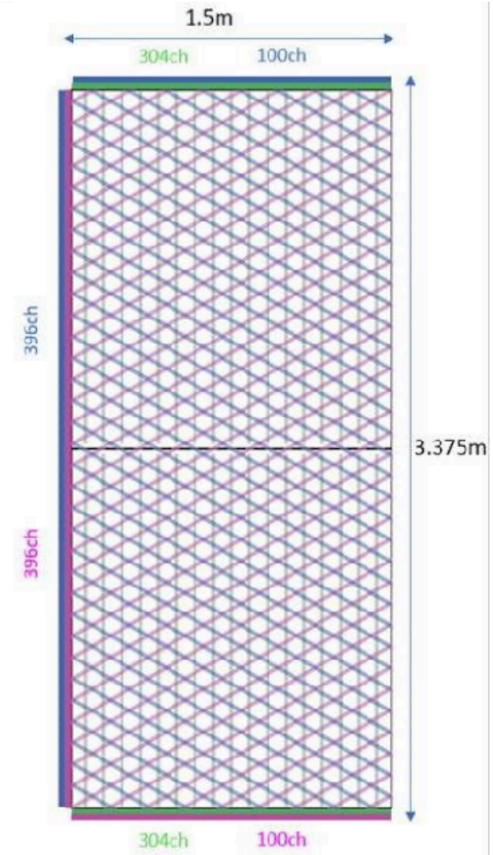
CVN for Vertical Drift

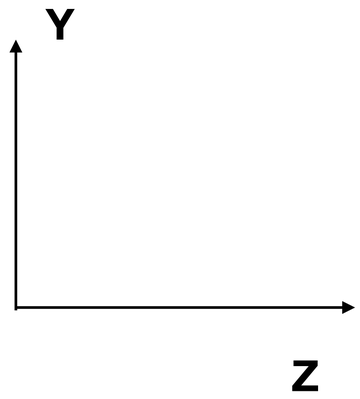
Nitish Nayak
13th Sept, 2021

Introduction

- Deciding on a Vertical Drift design is a top priority for DUNE moving forward
- Multiple options available :
  - 2-view : 1 induction, 1 collection — (0, 90) deg
  - 3-view : 2 induction, 1 collection — (0, 48, 90) deg (CDR geometry)
  - 3 view : 2 induction, 1 collection — (-30, 30, 90) deg
- 3 views gives us redundancy and in principle better 3D reconstruction
- Tricky to evaluate the physics case for each design without developing the whole reconstruction infrastructure







CVN

- Idea is to use CVN as a possible input to deciding which design might be better
  - Highly performant neutrino ID [https://arxiv.org/abs/2006.15052]
  - Just uses images of hit clusters as inputs -> downstream reco not necessary
  - Can help us quantify differences in performance between different designs
- Preliminary study on this in Oct 2020 by Sandro, Saul et al. :
  - Used horizontal drift simulation, trainings done for 1-view, 2-view, 3-view options by manually removing one of the view inputs each time
  - Interesting results showing small drop in performance across the trainings

DUNE CVN (views 0, 1, and 2)

	precision	Recall	F1 score	#events
CC ν_μ	**0.93**	**0.96**	**0.95**	**26108**
CC ν_e	**0.93**	**0.97**	**0.95**	**25665**
CC ν_τ	0.66	0.37	0.47	5813
NC	0.94	0.95	0.94	42382

Collection plane (view 2)

	precision	Recall	F1 score	#events
CC ν_μ	**0.91**	**0.94**	**0.92**	**26108**
CC ν_e	**0.90**	**0.94**	**0.92**	**25665**
CC ν_τ	0.59	0.26	0.36	5813
NC	0.91	0.93	0.92	42382

Induction planes (views 0 and 1)

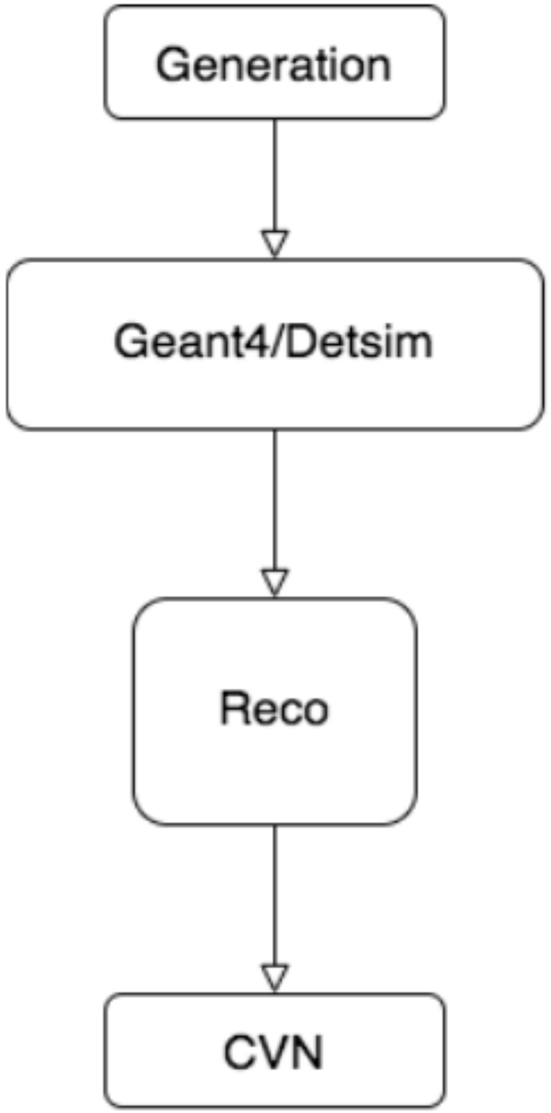
	precision	Recall	F1 score	#events
CC ν_μ	**0.91**	**0.95**	**0.93**	**26108**
CC ν_e	**0.90**	**0.95**	**0.92**	**25665**
CC ν_τ	0.59	0.27	0.37	5813
NC	0.92	0.93	0.92	42382

Induction plane, collection plane (views 1 and 2).

	precision	Recall	F1 score	#events
CC ν_μ	**0.91**	**0.95**	**0.93**	**26108**
CC ν_e	**0.90**	**0.95**	**0.92**	**25665**
CC ν_τ	0.58	0.31	0.40	5813
NC	0.92	0.92	0.92	42382

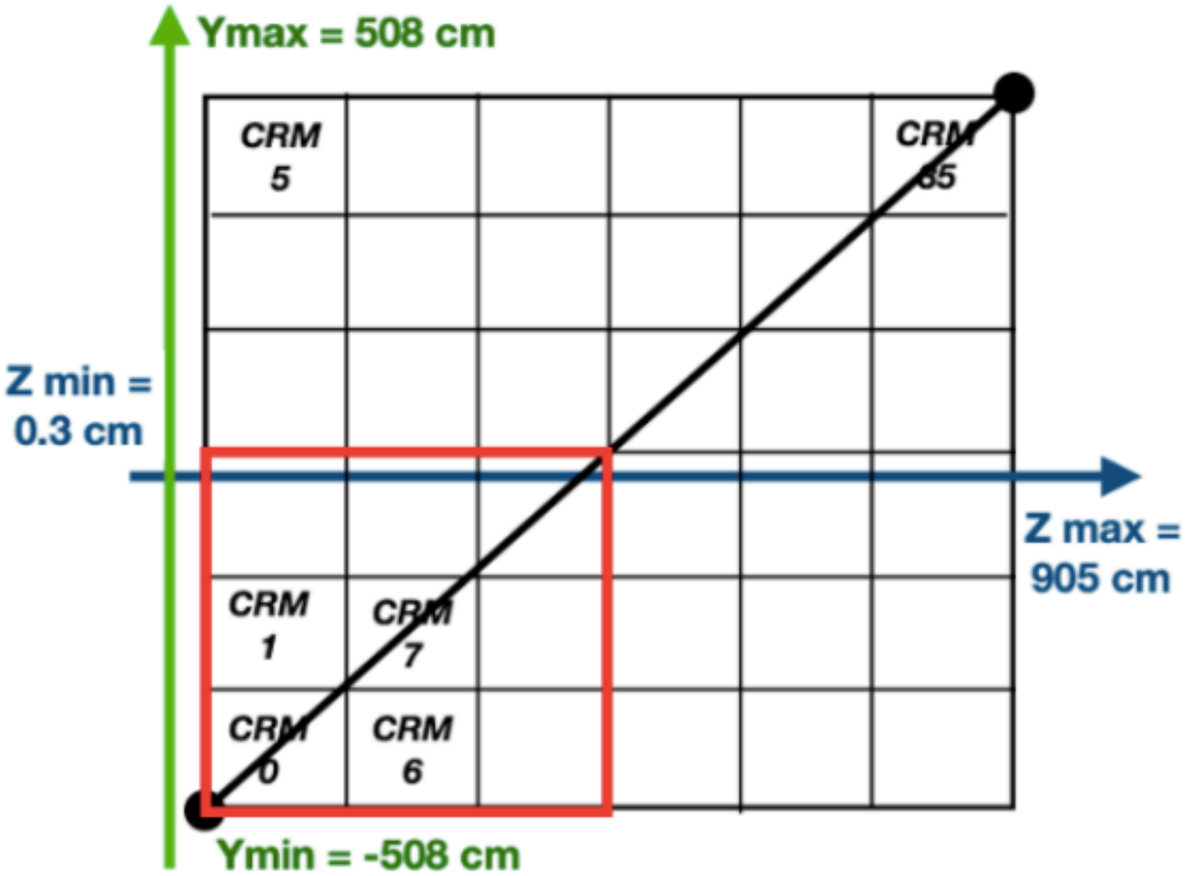
https://indico.cern.ch/event/864638/contributions/4041234/attachments/2132971/3592132/SP-EPNU-29oct20.pdf

VD Simulation



- fcls exist for the 2-view and CDR reference geometry — (0, 48, 90) deg
- For the other 3-view design, geometry workspace already created and wire-cell simulation already setup by Slavic/Haiwang
  - Made fcls for each of these steps
- All of these for the 1x6x6 VD geometry workspace (i.e 36 charge readout modules [CRMs] with one drift volume)
- Detsim uses json config in DUNEWireCell to produce RawDigit products
  - pgrapher/experiment/dune-vd/wcls-sim-drift-simchannel-3view30deg.json
- Worked out of Dom's refactor branch : feature/dbrailsf_refactor until recently
- Using another feature branch off of that for CVN-related development work (feature/bnayak_cvnvd)

VD Geometry



- 1x6x6 geometry workspace
  - 1 drift volume
  - 298 strips in 2 induction planes
  - 304 strips in 1 collection plane
- Checked numbering convention for CRMs in the wire-cell detsim vs larsoft and they're consistent

Reconstruction — Hit Finding

```
195 # gauss hit finder for VD module
196 dunevdfd_gaushitfinder: @local::gaus_hitfinder
197 dunevdfd_gaushitfinder.HitFinderToolVec.CandidateHitsPlane0.RoiThreshold: 6.0
198 dunevdfd_gaushitfinder.HitFinderToolVec.CandidateHitsPlane1.RoiThreshold: 0.6
199 dunevdfd_gaushitfinder.HitFinderToolVec.CandidateHitsPlane2.RoiThreshold: 0.6
200 dunevdfd_gaushitfinder.InitWidth: [6.0, 6.0, 6.0]
201 dunevdfd_gaushitfinder.AreaNorms: [13.25, 13.25, 13.25]
202 dunevdfd_gaushitfinder.MaxMultiHit: 4
203 dunevdfd_gaushitfinder.Chi2NDF: 50
204 dunevdfd_gaushitfinder.LongMaxHits: [ 25, 25, 25 ]
205 dunevdfd_gaushitfinder.LongPulseWidth: [ 10, 10, 10 ]
206 dunevdfd_gaushitfinder.PeakFitter:      @local::peakfitter_mrqdt
```

- Currently following Slavic's hit-finder talk for gaus-hit threshold parameters : https://indico.fnal.gov/event/50066/contributions/219793/attachments/145290/184914/vdfd_hitreco.pdf
  - Applied to new 3view30deg design
  - 1st induction plane has higher threshold to deal with pepper-noise issue (discussed previously, solved by new wire cell sig-proc parameters. New config should be available soon)

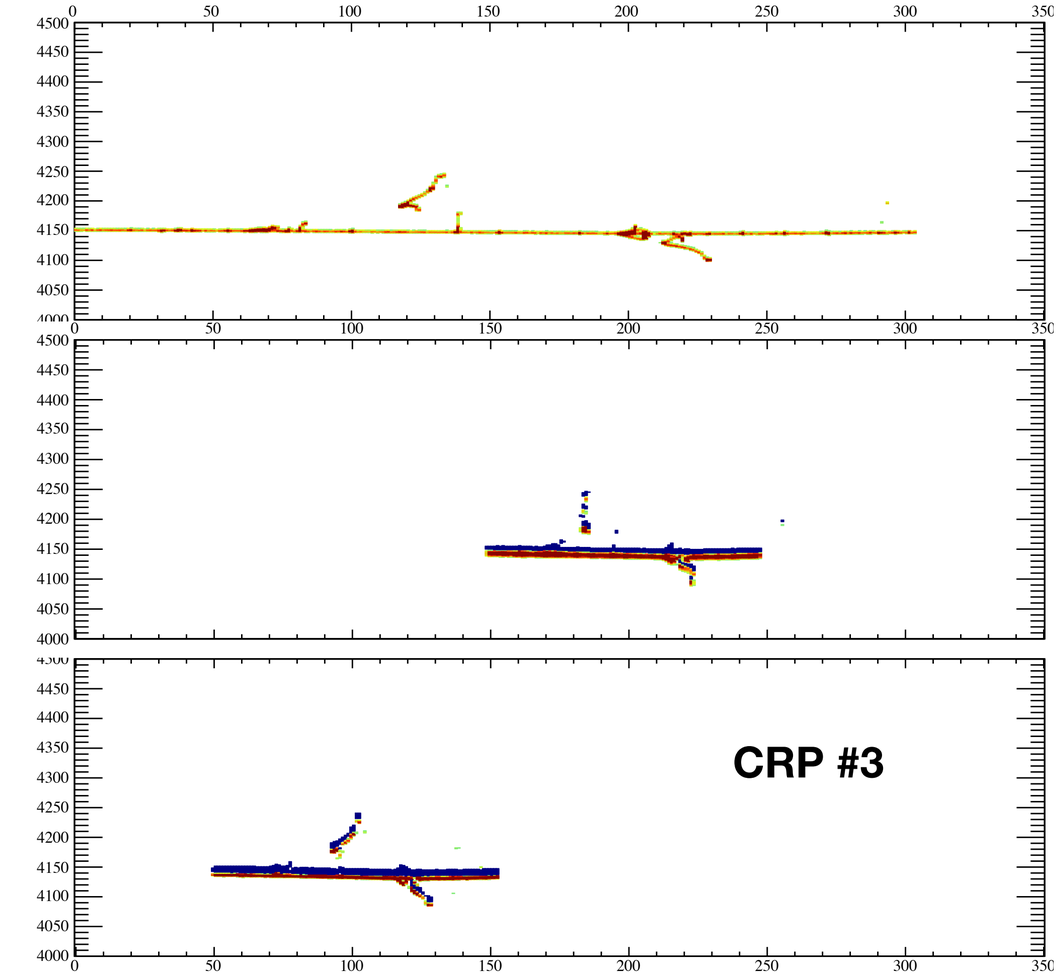
Reconstruction

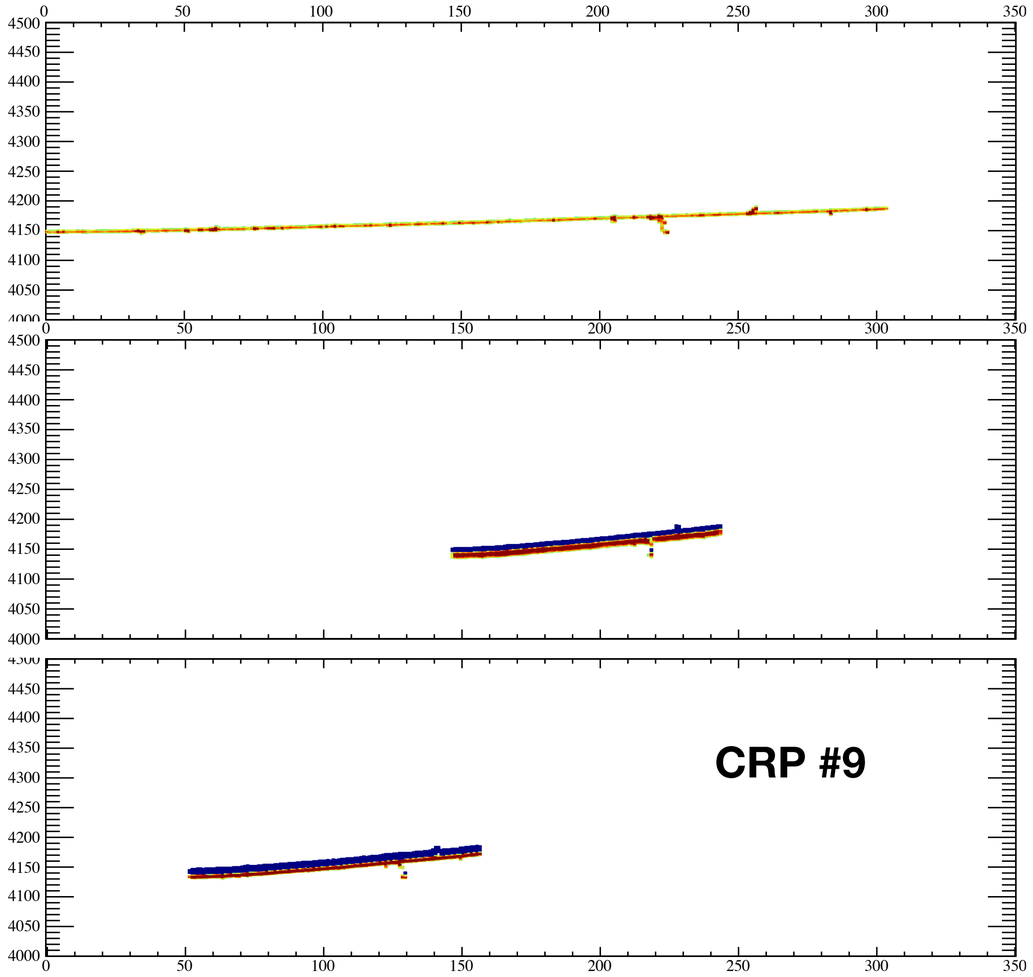
PROCESS NAME	MODULE LABEL..	PRODUCT INSTANCE NAME.....	DATA PRODUCT TYPE.....	.SIZE
SinglesGen..	generator.....	.....	std::vector<simb::MCTruth>.....	1
SinglesGen..	rns.....	.....	std::vector<art::RNGsnapshot>.....	1
SinglesGen..	TriggerResults	.....	art::TriggerResults.....	....-
G4.....	elecDrift.....	.....	std::vector<sim::SimChannel>.....	.3110
G4.....	rns.....	.....	std::vector<art::RNGsnapshot>.....	3
G4.....	IonAndScint...	.....	std::vector<sim::SimEnergyDeposit>.....	30783
G4.....	TriggerResults	.....	art::TriggerResults.....	....-
G4.....	largeant.....	.....	std::vector<simb::MCParticle>.....	.1744
G4.....	largeant.....	LArG4DetectorServicevolTPCActive	std::vector<sim::SimEnergyDeposit>.....	30783
G4.....	largeant.....	.....	art::Assns<simb::MCTruth,simb::MCParticle,sim::GeneratedParticleInfo>	.1744
detsim.....	tpcrawdecoder.	simpleSC.....	std::vector<sim::SimChannel>.....	32400
detsim.....	tpcrawdecoder.	daq.....	std::vector<raw::RawDigit>.....	32400
detsim.....	rns.....	.....	std::vector<art::RNGsnapshot>.....	0
detsim.....	TriggerResults	.....	art::TriggerResults.....	....-
Reco.....	TriggerResults	.....	art::TriggerResults.....	....-
Reco.....	wclsdatanfsp..	wiener.....	std::vector<recob::Wire>.....	32400
Reco.....	gaushit.....	.....	std::vector<recob::Hit>.....	.3920
Reco.....	gaushit.....	.....	art::Assns<recob::Wire,recob::Hit,void>.....	.3920
Reco.....	wclsdatanfsp..	gauss.....	std::vector<recob::Wire>.....	32400
Reco.....	rns.....	.....	std::vector<art::RNGsnapshot>.....	0

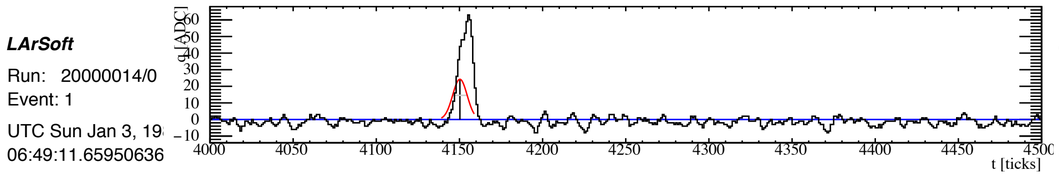
- Event dump after wire-cell signal processing (recob::Wire) + gaushit (recob::Hit)

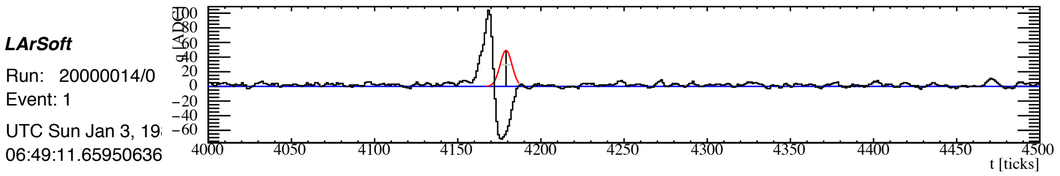
Event Displays

- Single fwd going muon









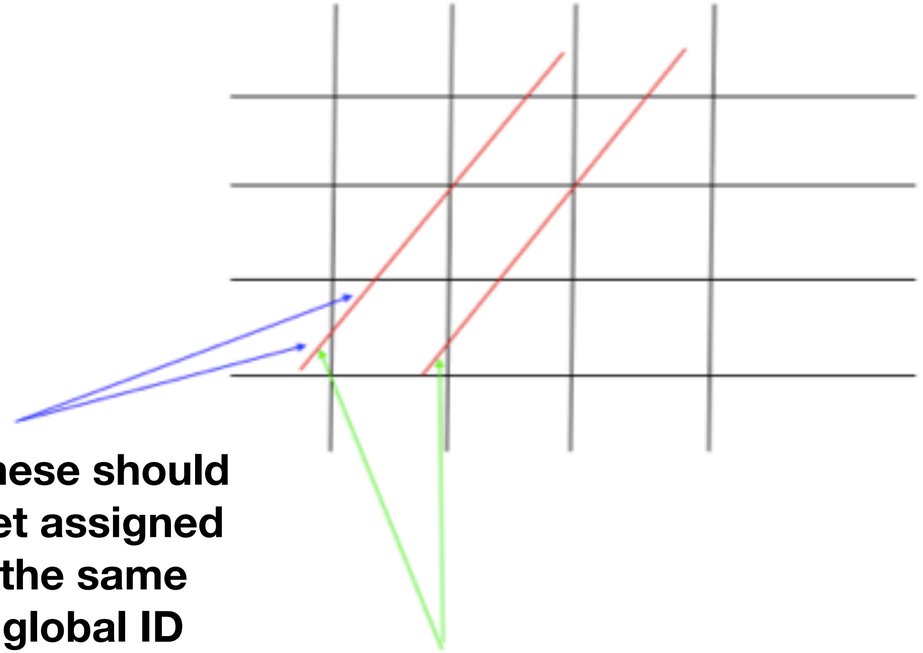
CVN

- Main module to create pixel map inputs from reconstructed files :
- Currently uses recob::Hit(s) from gaud-hit to populate 2D image of Wire ID vs TDC for each plane view
  - This is handled by the CVNMapper module
  - Takes into account events that cross multiple drift volumes as well (relevant for horizontal drift geometries, not so much for our scenario right now)
  - Calls a class `PixelMapProducer` that creates `PixelMap` objects to store in an art event
  - Workflow from there is fairly straightforward, uses a `CVNZLibMaker` module to convert these into compressed zip files (`.gz` format). Training data saved in separate txt files (`.info` format)
- CVNMapper fcl parameters :
  - WireLength : Set to 2880 for 1x2x6 Horizontal Drift workspace (max z-axis span)
  - TimeResolution : Set to 1600 — describes TDC range for pixel map (tdc_mean-1600, tdc_mean+1600)
  - TdcWidth : Set to 500 — condenses TDC ticks into 500 bins (Any reason for condensing 1600 to 500 specifically, rather than using an exact factor?)
  - For testing purposes, I'm not changing any of these currently except — TimeResolution to 1500.

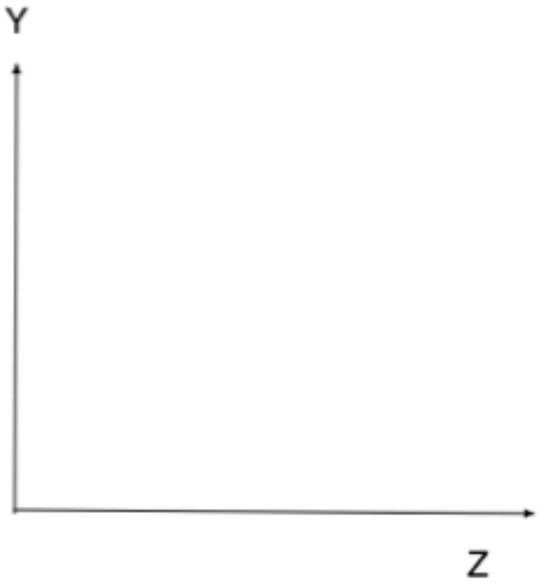
CVN

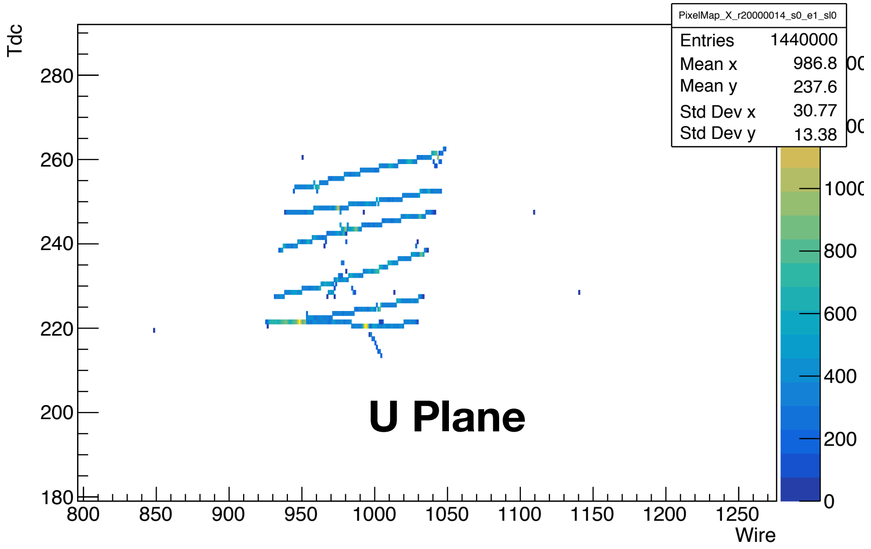
- PixelMapProducer tries to handle events crossing APAs by assigning a global wireID and time tick to each hit based on its location in the geometry
  - A bit complicated for horizontal drift since multiple drift volumes
- For our case, since there's only 1, things are somewhat easier
  - Global TDC = local TDC
- However, instead of multiple drift volumes we have multiple CRMs, each with a local strip "wireID"
- Need a method to assign a global wireID here as well
- Collection plane is easy. It's just :
  - Global wireID = local wireID + CRM_column*304
- Induction plane is a bit more complicated

Global Wire IDs

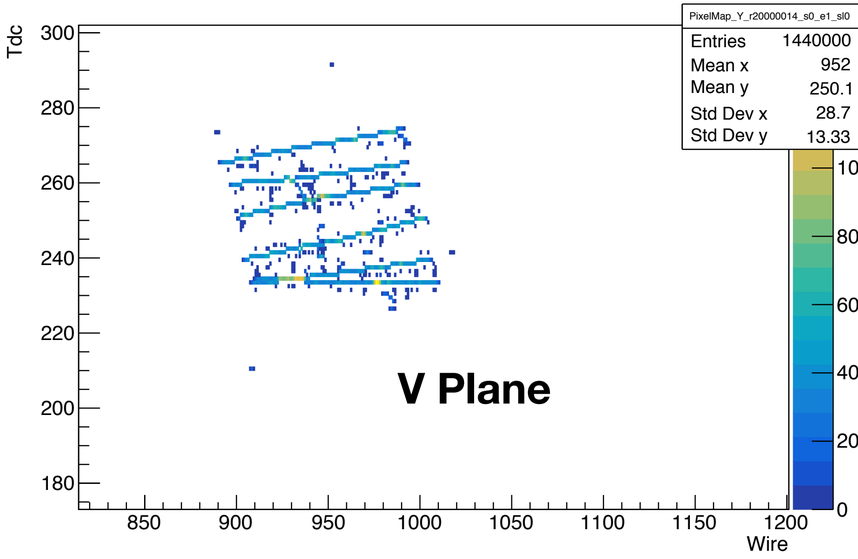


These should get assigned the same global ID



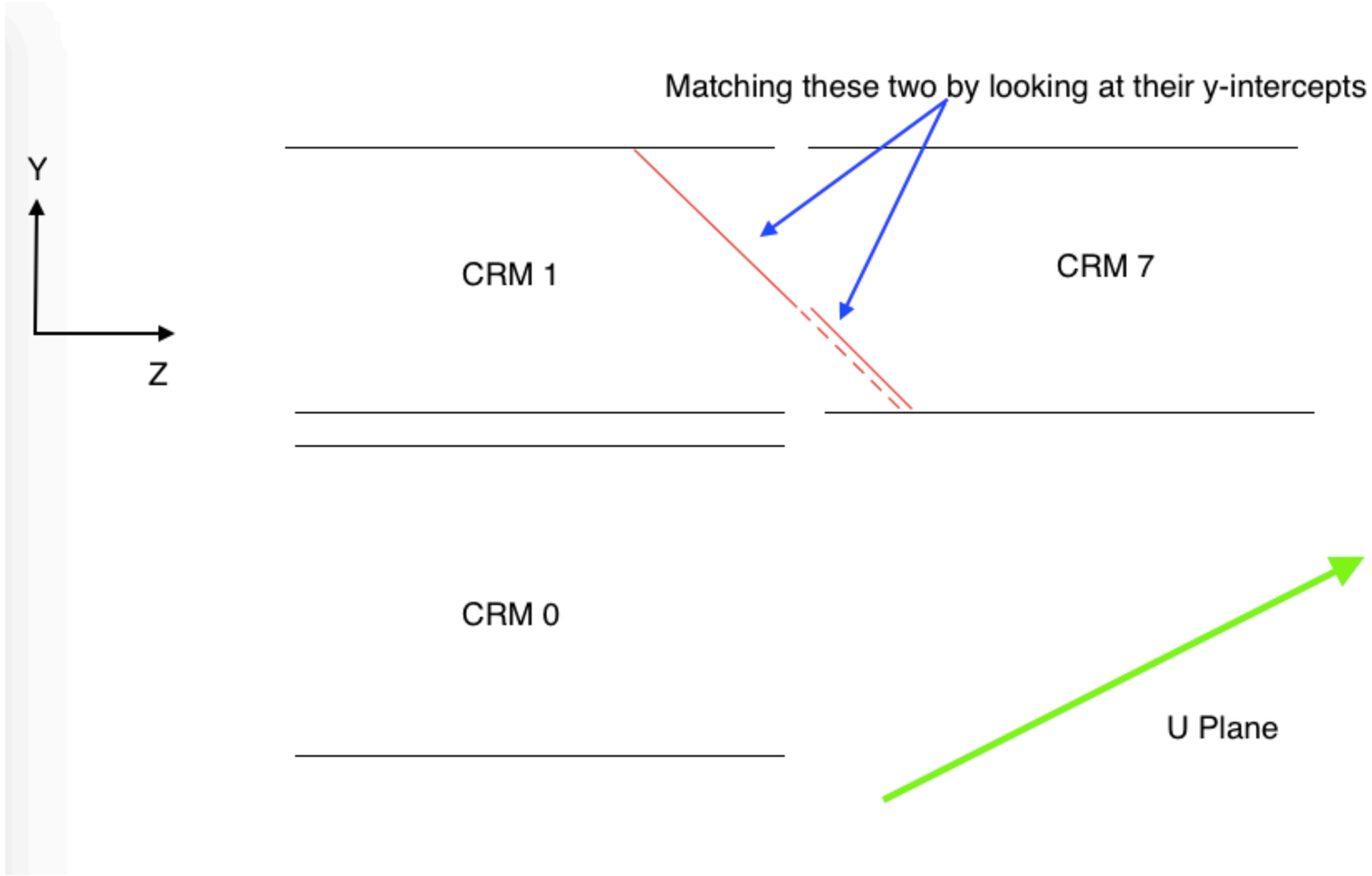


Parallel wires across CRMs have the same local WireID



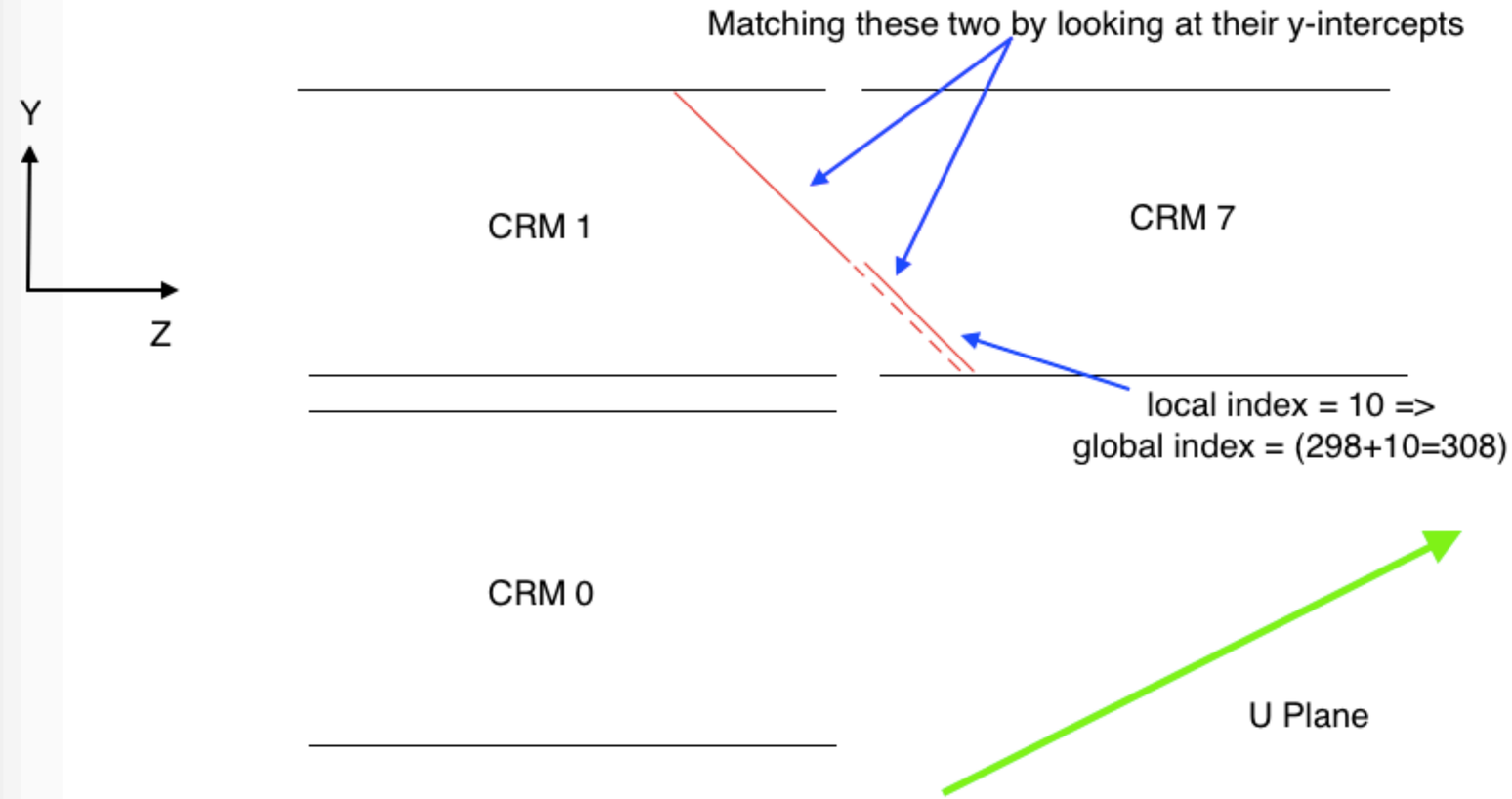
- Can’t do what is done for the collection plane.. otherwise you end up with pixel maps that look like these
- Have to be a bit smarter about matching wires across CRMs
  - Matching is not guaranteed/perfect in the design currently, but we can try to use some approximated version

Global Wire IDs



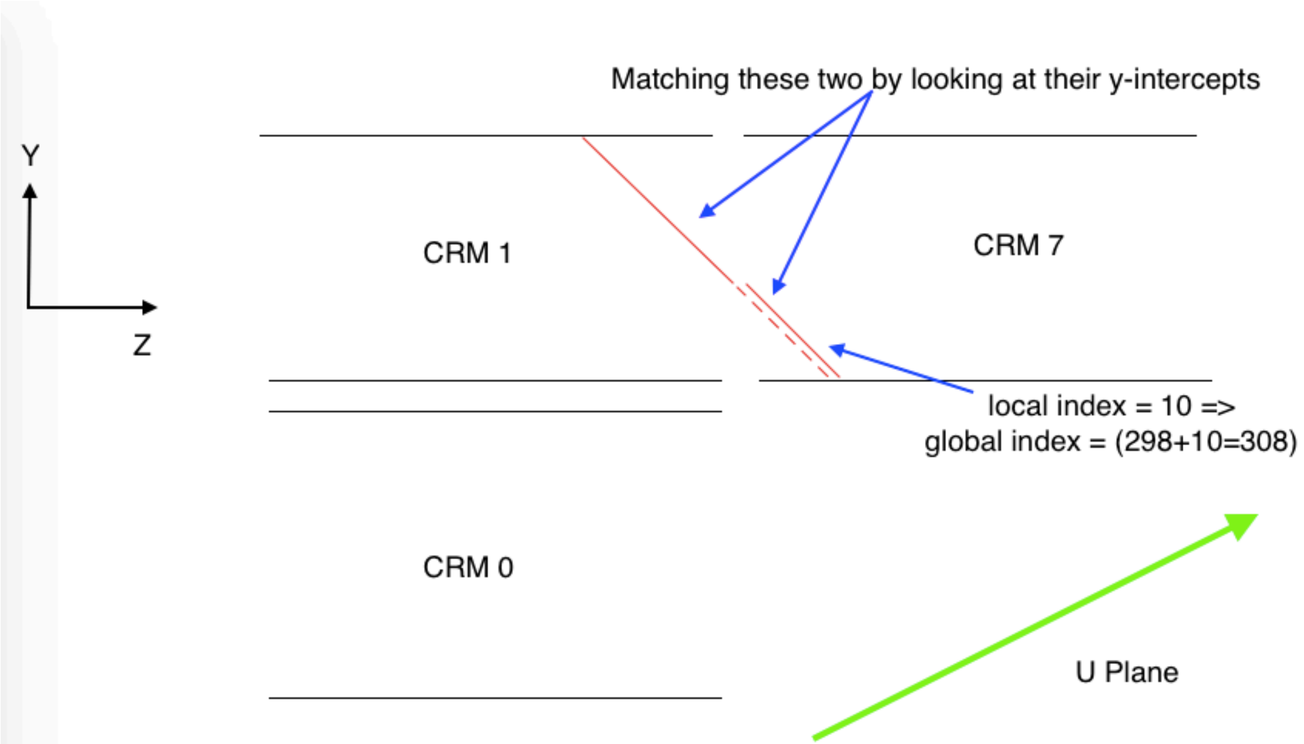
- illustration of how I'm matching wires across CRMs
- try to match every wire to a corresponding wire on a diagonal CRM
- matching done by looking at closest wire on diagonal CRM in terms of y-intercept
- intercept = (y0 +/- z0/sqrt(3))

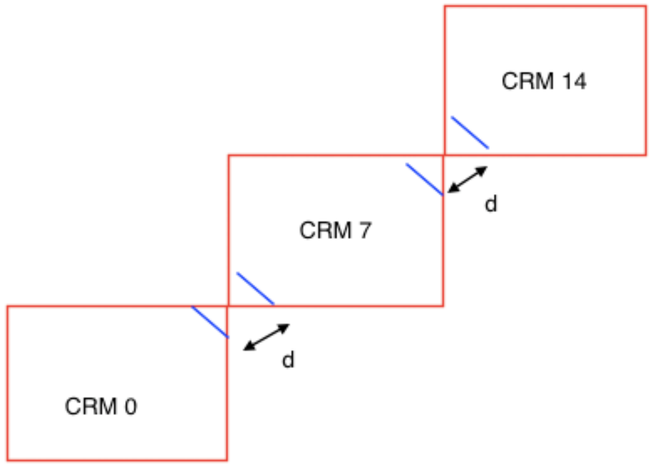
Global Wire IDs



- get local index of matched wire on diagonal CRM by counting wires from the start of the CRM
  - checked intercept spacing between neighbouring wires = 8.47 mm (from GDML). 8.47 = 7.335/cos(30deg)
  - 7.335mm = wire pitch in this geometry based on the 50L design tested at CERN (These numbers are in flux a little bit, but for now it doesn't matter)
- Get global index of matched wire from local index

Global Wire IDs

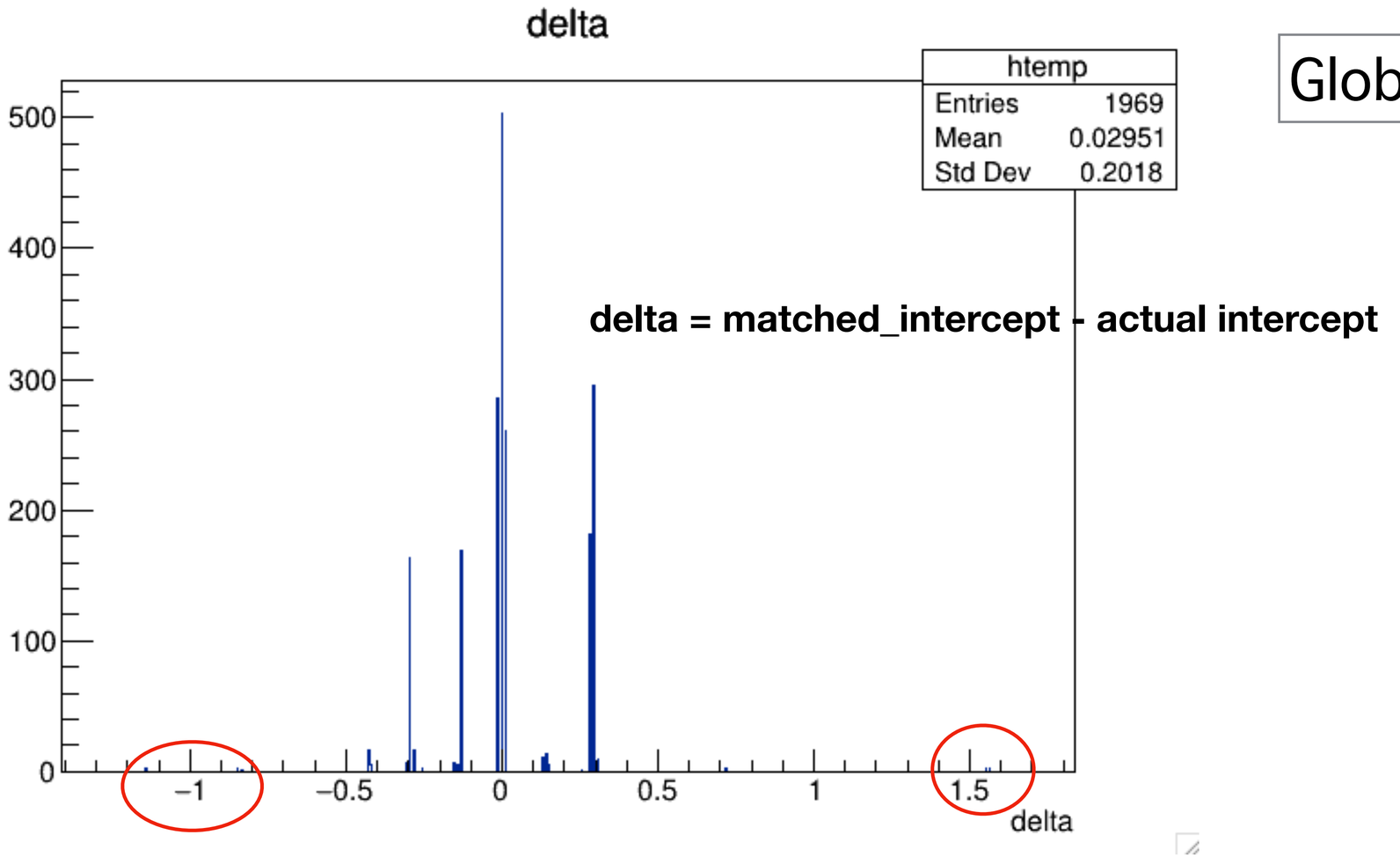




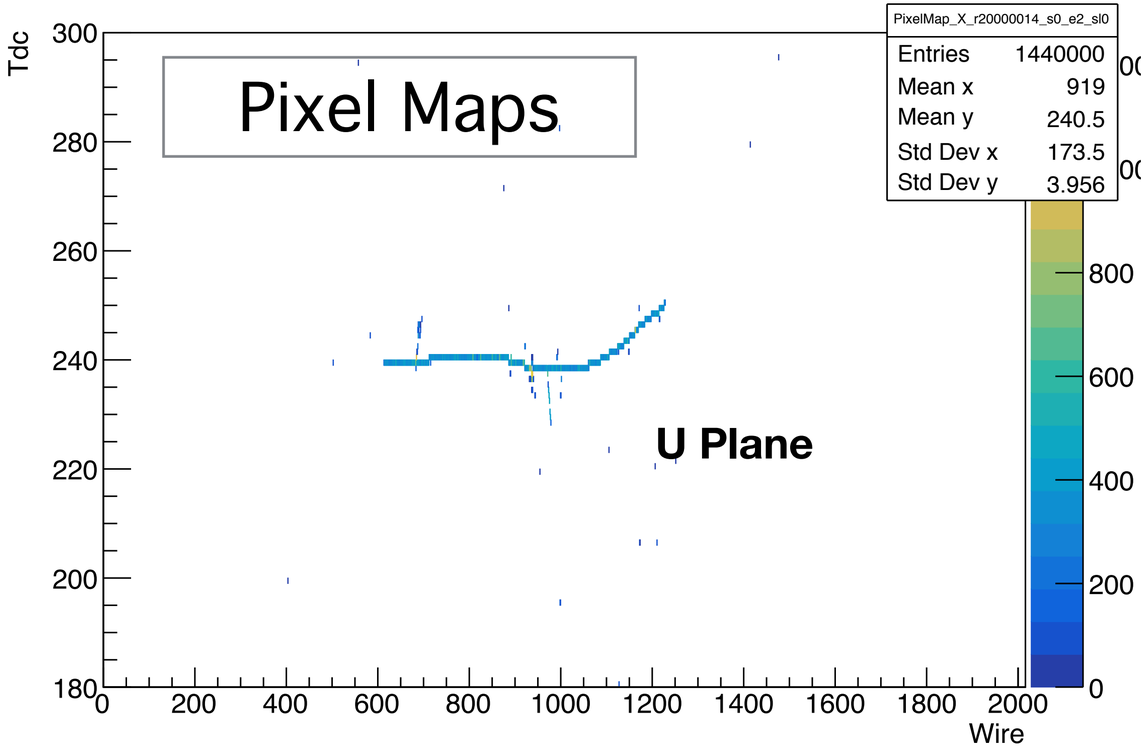
- For wires that project between two diagonal CRMs, I just assign it to the wire in one of the diagonal CRMs which it is closest to

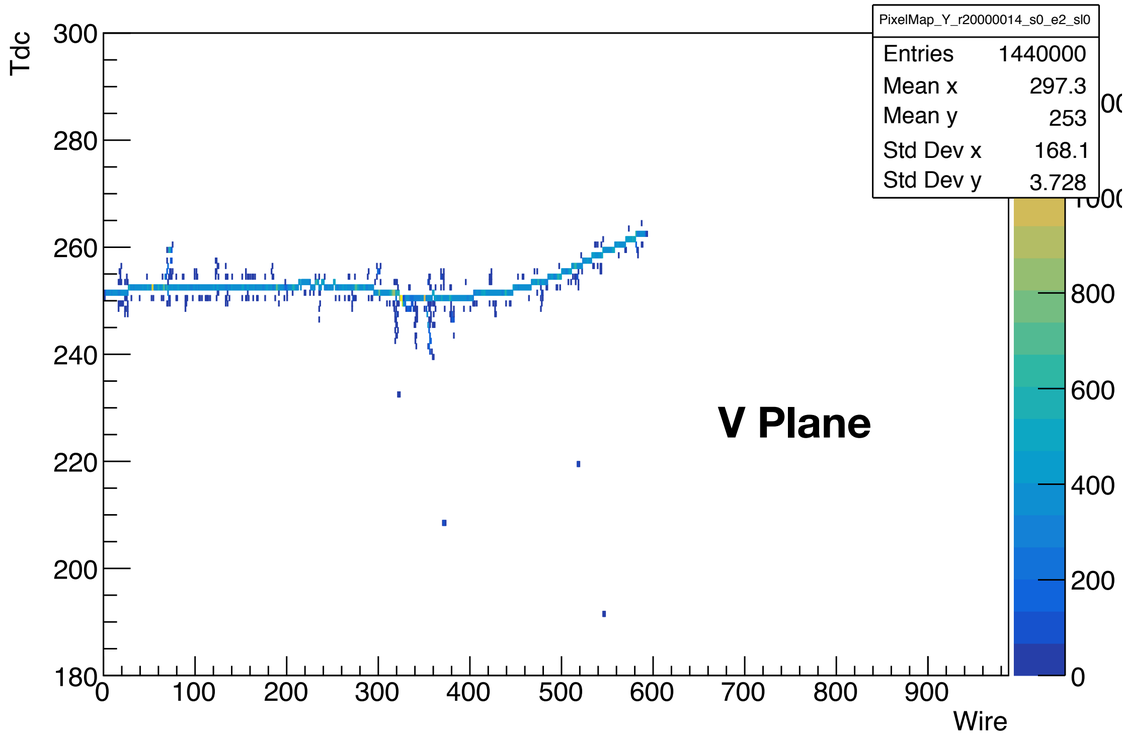
- Distance between these end wires across the diagonal CRMs alternate between d = 3.078cm and d = 1.712cm, so matching error is bounded by d/2

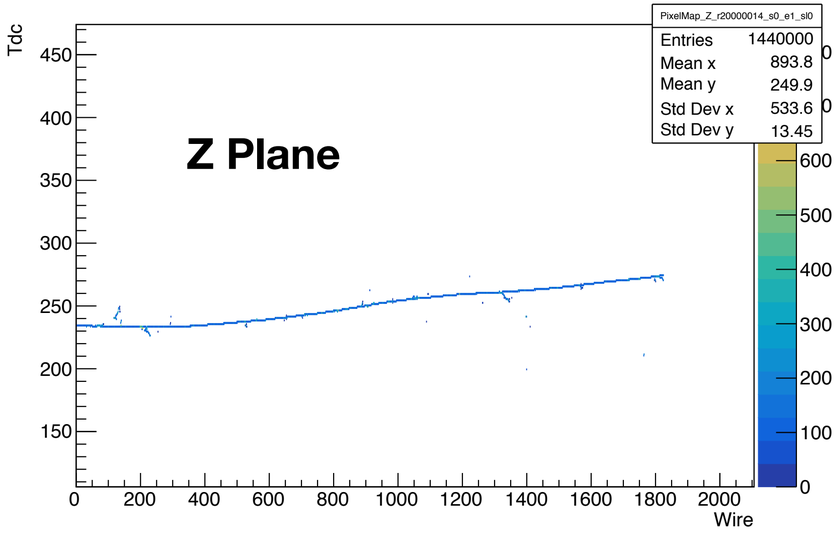
Global Wire IDs



- Most of the times, the matched wire intercept and the original intercept are the same
- The next frequent error is < 0.847/2 [~half a wire pitch]
- Rarely, error ~1.5 cm [twice the wire pitch] when wires project between diagonal CRMs
- Plans to make this more precise by maybe using variable binning pixel maps depending on where these wires project, rather than relying on a “matched” wire







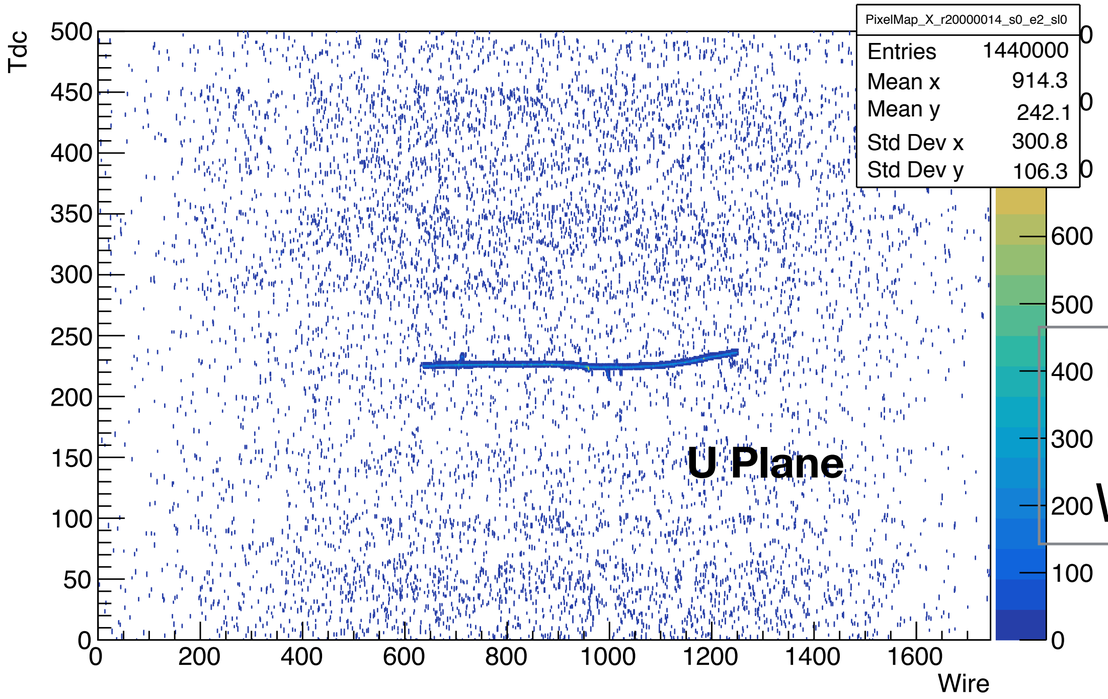
- New pixel maps look much more “reasonable”
- Caveats :
  - We see very prominent hairy features, especially in the V plane that doesn’t look physical
  - Haven’t studied performance of gaushit on the waveforms in a quantitative way for this design
  - In general, it seems more correct to use the waveforms directly in the pixel maps as inputs

Pixel Maps

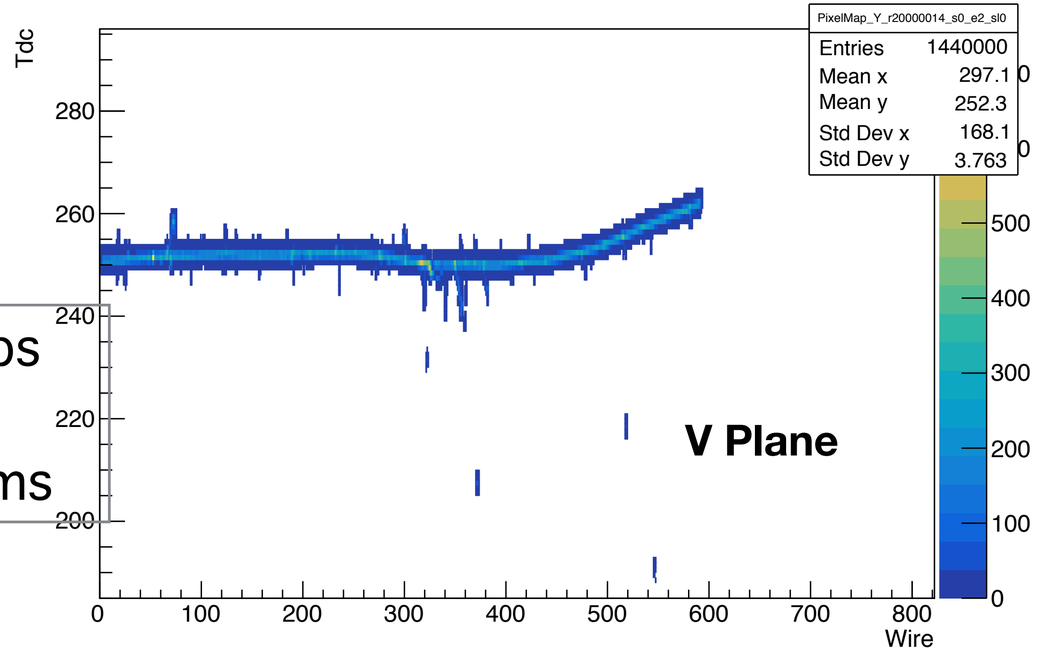
- Added infrastructure to make pixel maps using waveform inputs directly rather than using gaushits
  - New class : PixelMapWireProducer inside CVN that uses recob::Wire(s) instead of recob::Hits
  - Looping through wires, gathering the ROIs and filling the pixel map bins with the total charge content
  - This looping is not so expensive as it might seem since most recob::Wire(s) have no ROI content
  - Lives in feature/bnayak_cvnvd currently
- Keeping similar binning with what was done before for horizontal drift, but easy to change (=6 tdc ticks/bin)
- Goes through the wire-cell signal processing step : 2D deconv step + gaussian filter

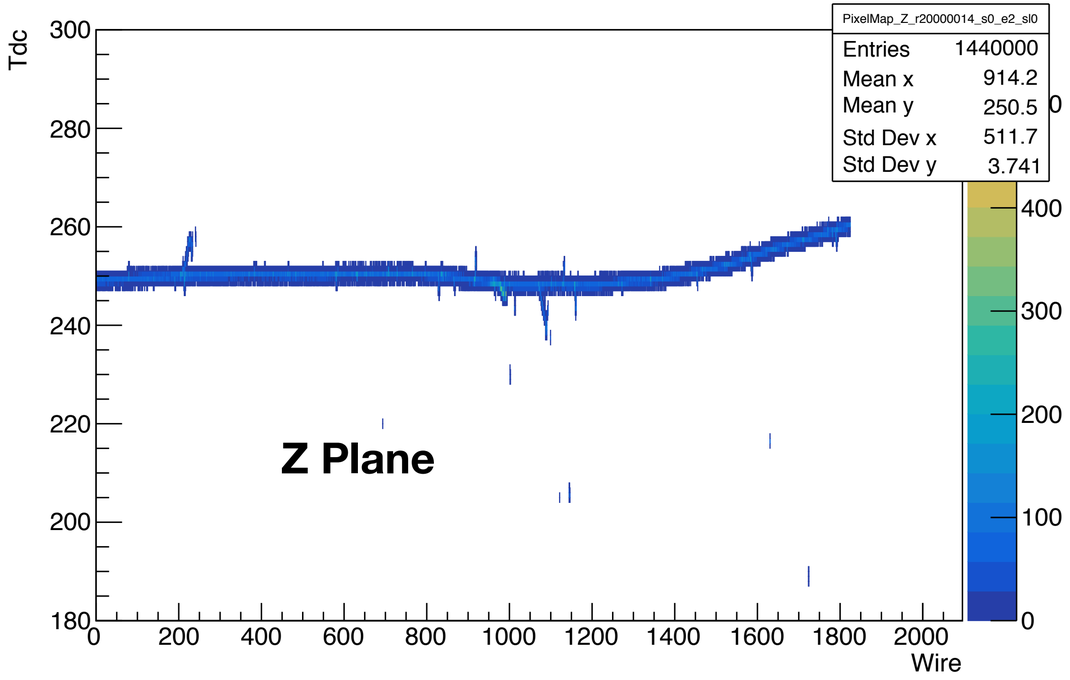
```
38 standard_cvnmapper_wire:
39 {
40   module_type:         CVNMapperWire
41   #=================
42   HitsModuleLabel:    "wclsdatanfsp:gauss"
43   ClusterPMLabel: "cvnmap"
44   MinClusterHits: 100
45   TdcWidth:       500
46   WireLength:     2880 #Unwrapped collection view max (6 x 480
47   TimeResolution: 1600
48   UnwrappedPixelMap: 1
49   Threshold: 0
50 }
```

- CVN/art/CVNMapper.fcl



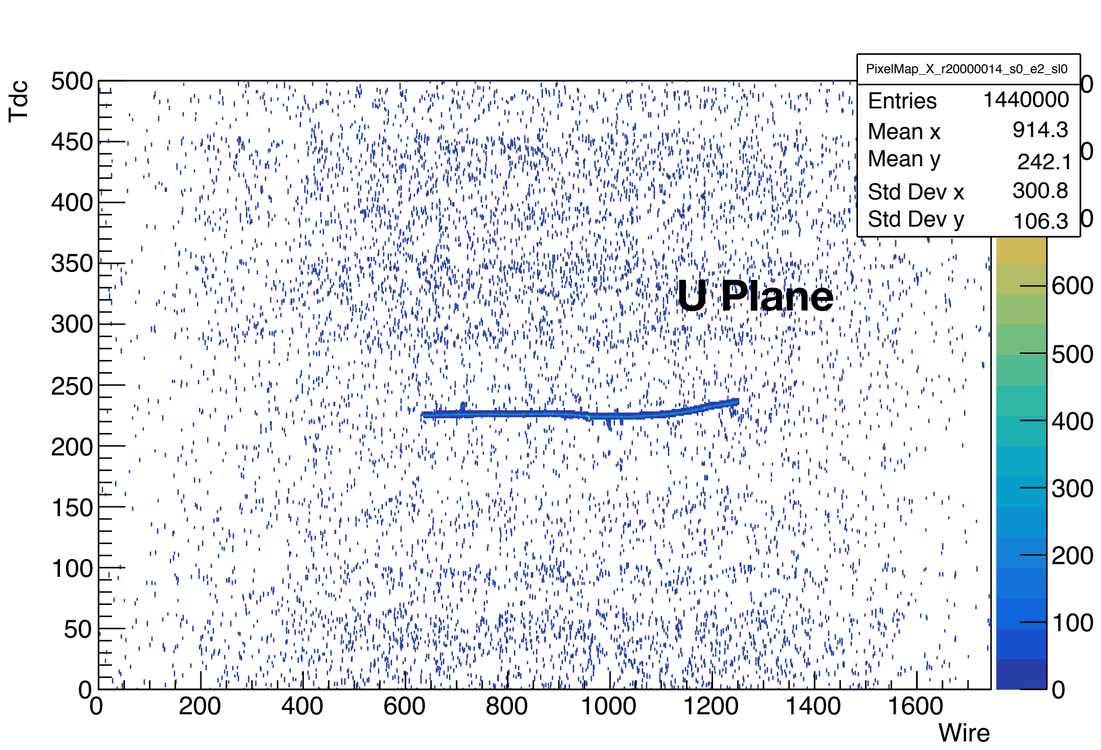
Pixel Maps using Waveforms

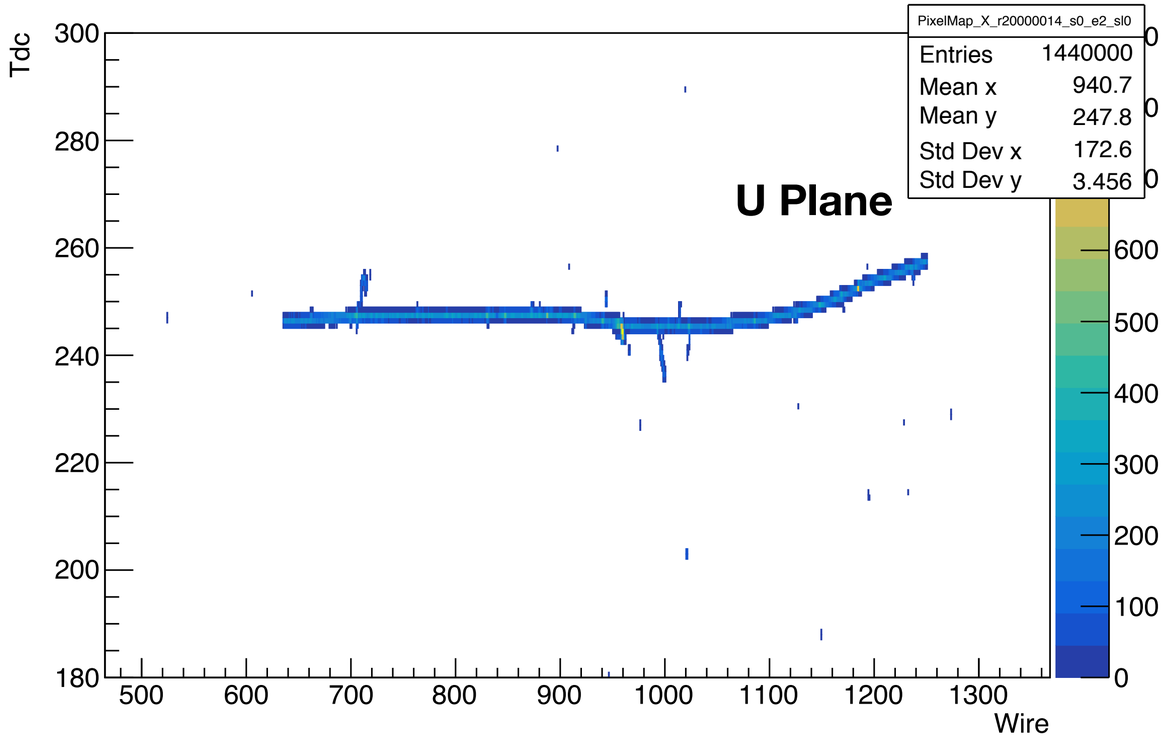




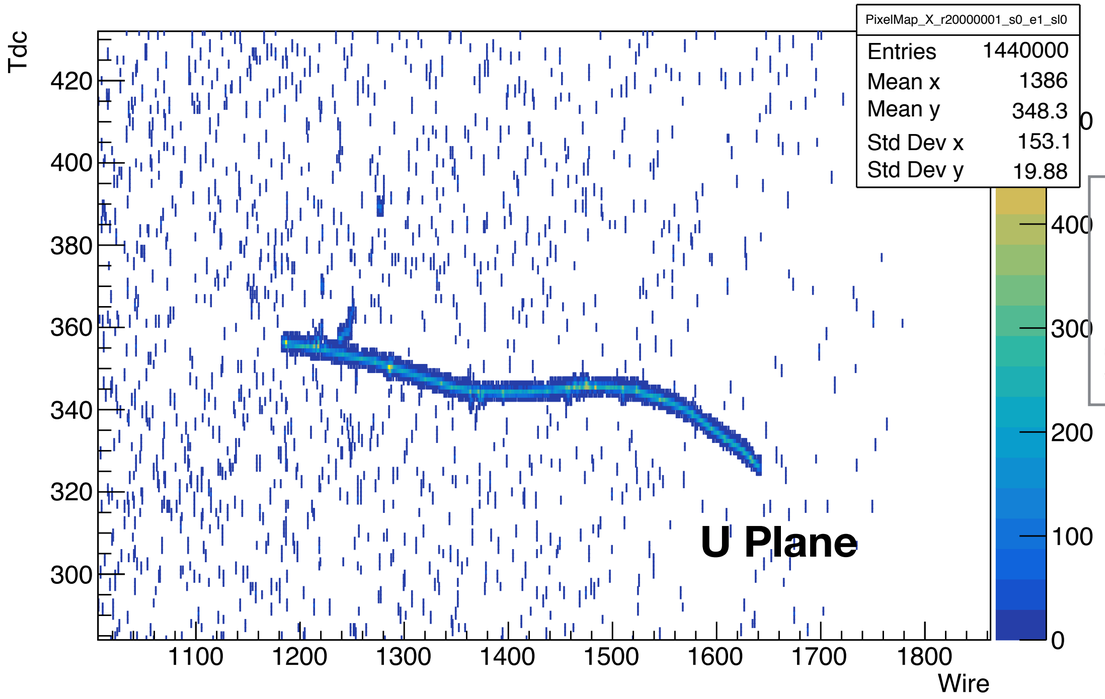
- Looks smoother than gaushit pixel maps
- U plane is very noisy, this is the same issue Slavic mentioned in FD sim/reco and we discussed last week —> should be good with new config (with more appropriate electronic response parameters in sigproc chain)
- Hairiness (in V plane esp.) still exists but doesn't look as bad

Pixel Maps using Waveforms

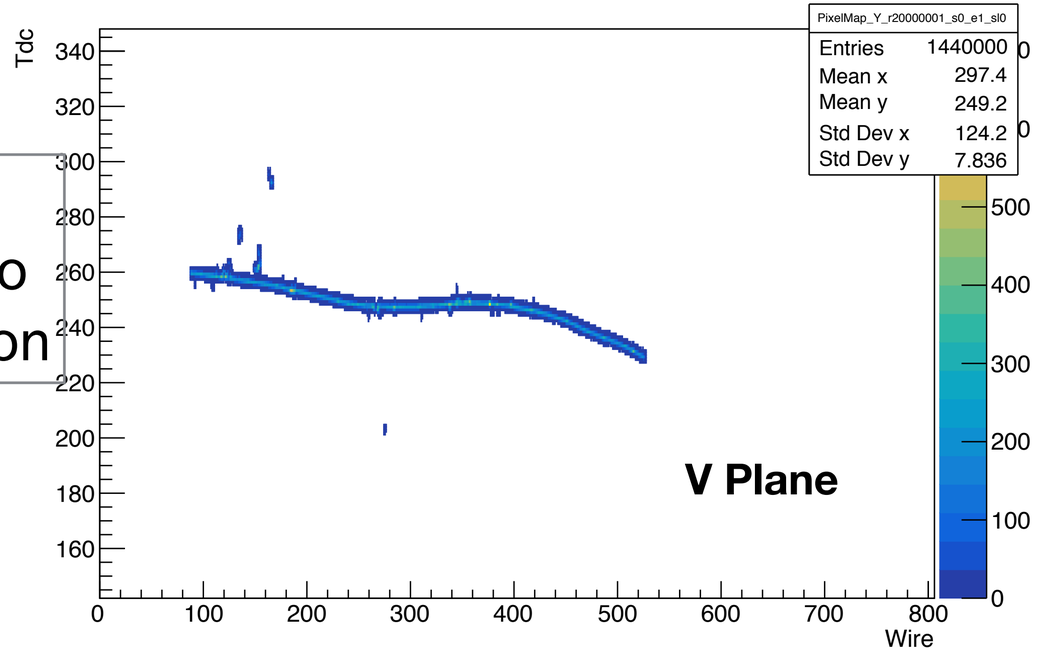


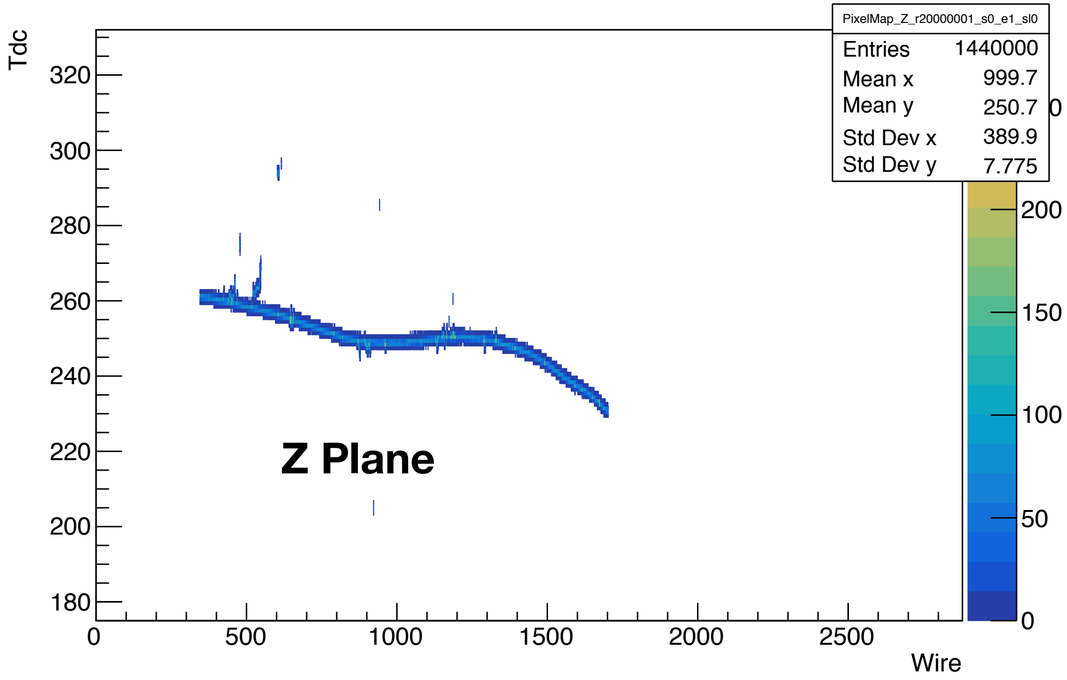


- Added a knob for a charge threshold in PixelMapWireProducer as well
- Need a threshold of 6 to get rid of all the noisy hits in the U plane with old config



Using Neutrino Simulation





- Dom mentioned on slack that the neutrino simulation using dk2nu flux files was up and running
- Tested the pixel maps here as well, seems to be working
- pixel maps stored as zlib-compressed outputs (.gz extension) with truth info stored in .info files as text output
- Once we generate a sizeable sample, can copy it to our workstations/cluster and train

Summary

- CVN would be very useful in testing the design choices currently for the VD
- Workflow seems to be in place for training CVN on VD geometries
  - both for waveform inputs and gaushit inputs
- Still a few things to clean up maybe :
  - Could do things more precisely with the stitching algorithm using matching strip intercepts
  - Probably some other intermediate steps/modifications to use CVN training scripts on these `.gz` and `.info` outputs
- Logistical questions :
  - Should we start to look at producing a larger training/testing sample on our own?
  - Something for production? Possible timeline/schedule etc?