

THE MicroBooNE Event Display

Tutorial at: https://cdcvs.fnal.gov/redmine/projects/uboone-physics-analysis/wiki/Gallery_Event_Display_-_How_To

When first setting up:

1. (go into SL7 container)

```
source /nashome/i/imawby/interactiveContainer_uboone.sh
```

2. Clone EVD dir

```
git clone https://github.com/davidc1/gallery-framework UBEVD
```

3. replace config/setup.sh with

```
/exp/uboone/app/users/imawby/UBEVD/config/setup.sh  
source config/setup.sh
```

4. Build (you only have to do this once)

```
cd $GALLERY_FMWK_BASEDIR  
make
```

To run:

1. Change the resolution of your vnc so the EVD window fits, type this into the uboonegsvm

```
xrandr -s 1600x1200
```

2. (go into SL7 container)

```
source /nashome/i/imawby/interactiveContainer_uboone.sh
```

3. source the setup script

```
source config/setup.sh
```

4. Run event display on reco2 files!

```
evd.py -T <path_to_file>
```

Isobel's Hacked Together MicroBooNE Event Display

This takes the form of a Jupyter notebook which runs on files created by my 'Visualisation analyser'. The intended workflow for this is:

reco2 files of interest -> run analyser -> move output files to computer with jupyter -> run notebook

DISCLAIMER: I pulled this together in the last few days, and am pretty pants at python so apologies if my notebook makes you sad. If something doesn't work let me know, it should be easy to fix. I'd appreciate any feedback too!

The Analyser

You can either access the analyser via 1) setting up my local larsoft build on the uboonegpvms or 2) moving my analyser into your local larsoft build

Option 1:

1. source my local larsoft build setup script

```
source /exp/uboone/app/users/imawby/larsoft_olderVersion/setup.sh
```

2. run analyser on events

```
lar -c run_VisualiseSlice.fcl <PATH_TO_YOUR_EVENTS>
```

You should then end up with a file called **reco_stage_2_hist.root** (sorry I didn't change the default name...) that's the one with the tree in!

Option 2:

1. Git clone my github repository somewhere in your uboonegpvm area. This repo contains a directory for the analyser and the jupyter notebook (we'll come to the latter soon). Move the analyser directory into your local build of ubana and build.

```
git clone https://github.com/imawby/WarwickuBooNEWorkshop24
```

```
mv EVDDir srcs/ubana/ubana/
```

open the CMakeLists.txt that lives inside srcs/ubana/ubana/ and add EVDDir to the list of directories

move to a build machine, cross those fingers and build

ASIDE: If you don't have ubana in your local larsoft build:

```
cd scrs
```

Event Display Options

mrbgubana

ups active (then find the version of ubana that is currently set up)

git checkout tags/PUT_VERSION_HERE -b AWESOME_BRANCH_NAME

move to a build machine, cross those fingers and build

2. run analyser on events

lar -c run_VisualiseSlice.fcl <PATH_TO_YOUR_EVENTS>

You should then end up with a file called **reco_stage_2_hist.root** (sorry I didn't change the default name...) that's the one with the tree in!

The Isobel's Hacked Together MicroBooNE Event Display

IMPORTANT: This needs to happen on a computer in which you can open a jupyter notebook. Can this happen on the uboonegpvms? I don't know. If you do know, put me out of my misery. I usually move over the ana files to my laptop and run things from there. Do this with a classic scp e.g.

scp

imawby@uboonegpvm02.fnal.gov:/exp/uboone/app/users/imawby/larsoft_olderVersion/junk/reco_stage_2_hist.root ./

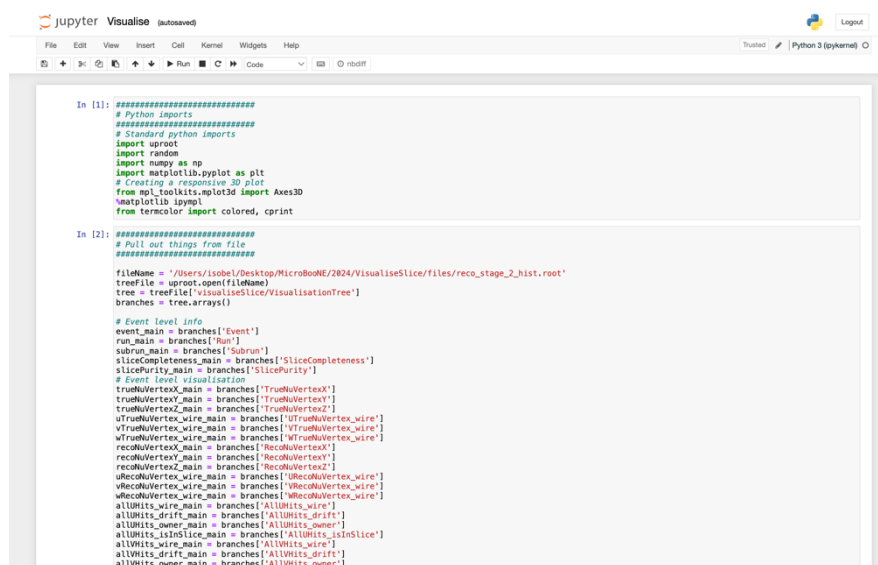
1. ON AN APPROPRIATE MACHINE, git clone my github repository. This repo contains a directory for the analyser and the jupyter notebook (we only care about the latter in this section).

git clone https://github.com/imawby/WarwickuBooNEWorkshop24

2. Go into this directory and type

jupyter notebook

3. Click on the 'Visualise.ipyn' and you should see this beauty



```
In [1]: #####
# Python imports
#####
# Standard python imports
import uproot
import random
import numpy as np
import matplotlib.pyplot as plt
# creating a responsive 3D plot
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt
from termcolor import colored, cprint

In [2]: #####
# Pull out things from file
#####

fileName = '/Users/isobel/Desktop/MicroBooNE/2024/VisualiseSlice/files/reco_stage_2_hist.root'
treeFile = uproot.open(fileName)
tree = treeFile['visualiseSlice/visualisationTree']
branches = tree.arrays()

# Event level info
event_main = branches['Event']
run_main = branches['Run']
subrun_main = branches['Subrun']
sliceCompleteness_main = branches['SliceCompleteness']
slicePurity_main = branches['SlicePurity']

# Event level visualisation
TrueNUVertex_main = branches['TrueNUVertex']
TrueNUVertex_wire_main = branches['TrueNUVertex_wire']
TrueNUVertex_drift_main = branches['TrueNUVertex_drift']
uTrueNUVertex_wire_main = branches['uTrueNUVertex_wire']
vTrueNUVertex_wire_main = branches['vTrueNUVertex_wire']
wTrueNUVertex_wire_main = branches['wTrueNUVertex_wire']
recoNUVertex_main = branches['RecoNUVertex']
recoNUVertex_wire_main = branches['RecoNUVertex_wire']
uRecoNUVertex_wire_main = branches['uRecoNUVertex_wire']
vRecoNUVertex_wire_main = branches['vRecoNUVertex_wire']
wRecoNUVertex_wire_main = branches['wRecoNUVertex_wire']
allUnits_wire_main = branches['AllUnits_wire']
allUnits_drift_main = branches['AllUnits_drift']
allUnits_owner_main = branches['AllUnits_owner']
allUnits_wireSlice_main = branches['AllUnits_wireSlice']
allUnits_wire_main = branches['AllUnits_wire']
allUnits_drift_main = branches['AllUnits_drift']
allUnits_owner_main = branches['AllUnits_owner']
```

Event Display Options

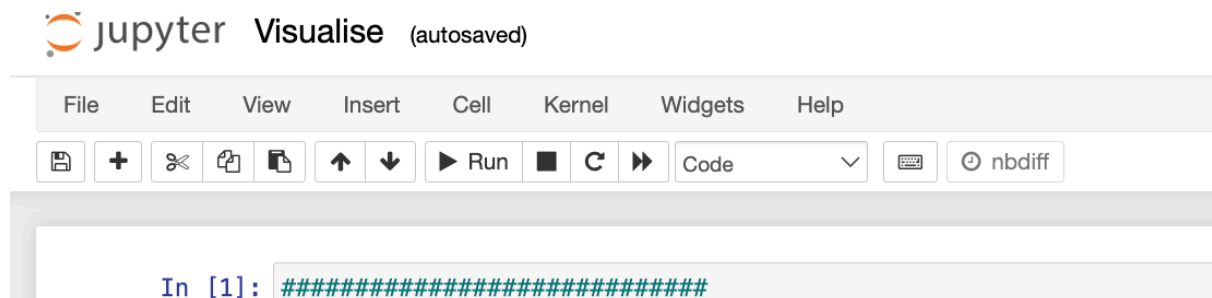
4. Put the name of your file here

```
#####  
# Pull out things from file  
#####  
  
fileName = '/Users/isobel/Desktop/MicroBooNE/2024/VisualiseSlice/files/reco_stage_2_hist.root'  
treeFile = uproot.open(fileName)  
tree = treeFile['visualiseSlice/VisualisationTree']  
branches = tree.arrays()
```

5. Put the index of the event you want to see here

```
spacePointsZ_main = branches['SpacePointsZ']  
spacePointsZ_main = branches['SpacePointsZ']  
  
In [3]: #####  
# Index of event in file to view  
#####  
  
iEvent = 3  
  
In [4]: #####  
# Get event event
```

6. You're ready to go... Hit the fast forward arrow to reset and run the whole notebook again



The screenshot shows the Jupyter Visualise interface. At the top, there is a logo for 'jupyter Visualise (autosaved)'. Below the logo is a menu bar with options: File, Edit, View, Insert, Cell, Kernel, Widgets, Help. Underneath the menu bar is a toolbar with icons for saving, adding cells, undo, redo, up/down arrows, a play button labeled 'Run', a square stop button, a refresh button, and a fast forward button. There is also a dropdown menu currently set to 'Code' and a keyboard icon. Below the toolbar is a code cell with the text 'In [1]: #####'.

7. Let me just give you a quick description of the things you'll see.. so first you'll want to check the run/subrun/event are those of the event that you actually care about... You'll be told that in this section. Here you're also told some reco properties, how many reco particles are in the reco'd nu slice, and the completeness/purity of that slice

```
In [7]: print('-----')  
print('--- EVENT INFO ---')  
print('-----')  
  
print('Event:', event_event)  
print('Run:', run_event)  
print('Subrun:', subrun_event)  
print('')  
print('Flash match slice completeness:', str(int(round(sliceCompleteness_event * 100, 0))) + '%')  
print('Flash match slice purity:', str(int(round(slicePurity_event * 100, 0))) + '%')  
print('# reco particles:', nParticles)  
  
-----  
--- EVENT INFO ---  
-----  
  
Event: 1272  
Run: 14271  
Subrun: 25  
  
Flash match slice completeness: 98%  
Flash match slice purity: 72%  
# reco particles: 6
```

Event Display Options

8. You'll then see all the true hits of the neutrino interaction. If the reco was perfect, this is the output to expect. So you should use this display as a benchmark of the reco. The hits are coloured by the PDG of the particle that owns them:

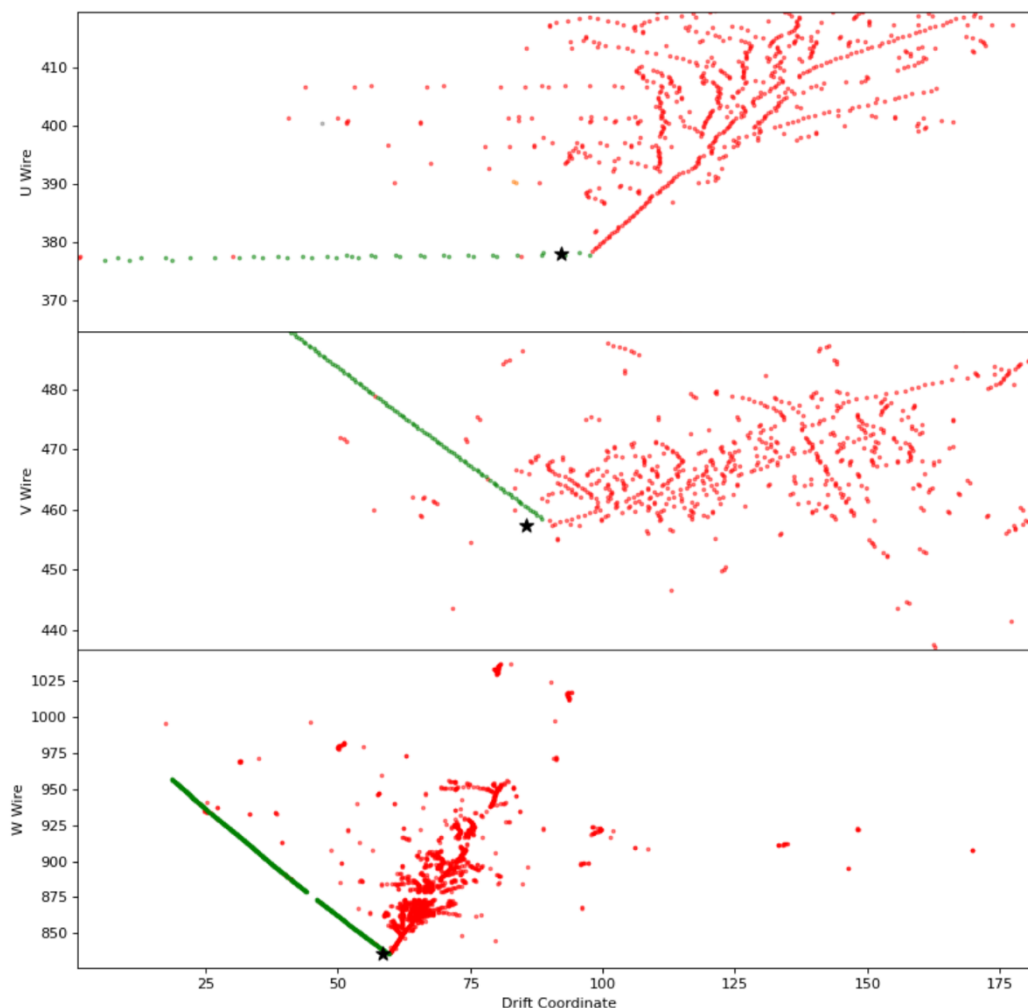
red – electron, orange – photon, green – proton, blue – muon, pink – pion, kaon - brown

Any particle not in this list will be drawn as grey. If you need to add a particle, find this line in the code and drop your particle in with some colour (not grey)

```
pdgColours_graph = {13 : 'blue', 11 : 'red', 2212 : 'green', 211 : 'pink', 22 : 'tab:orange', 321 :  
'tab:brown'}
```

The black star shows the true neutrino vertex. To get true positions to match to reco hits you need to apply all sorts of corrections (SCE, trigger offsets, wirecell-pandora differences). In this display I've only applied SCE (I couldn't get the trigger offset to work). So you might have to mentally shift the start in the x direction.

The x-axis is the drift coord, and the y the 'wire-coord'. This is the way it should be ;) In the unzoomed case, the hits will align in the x-direction (because the drift coord is common between views). This helps identify particles/features across the views.



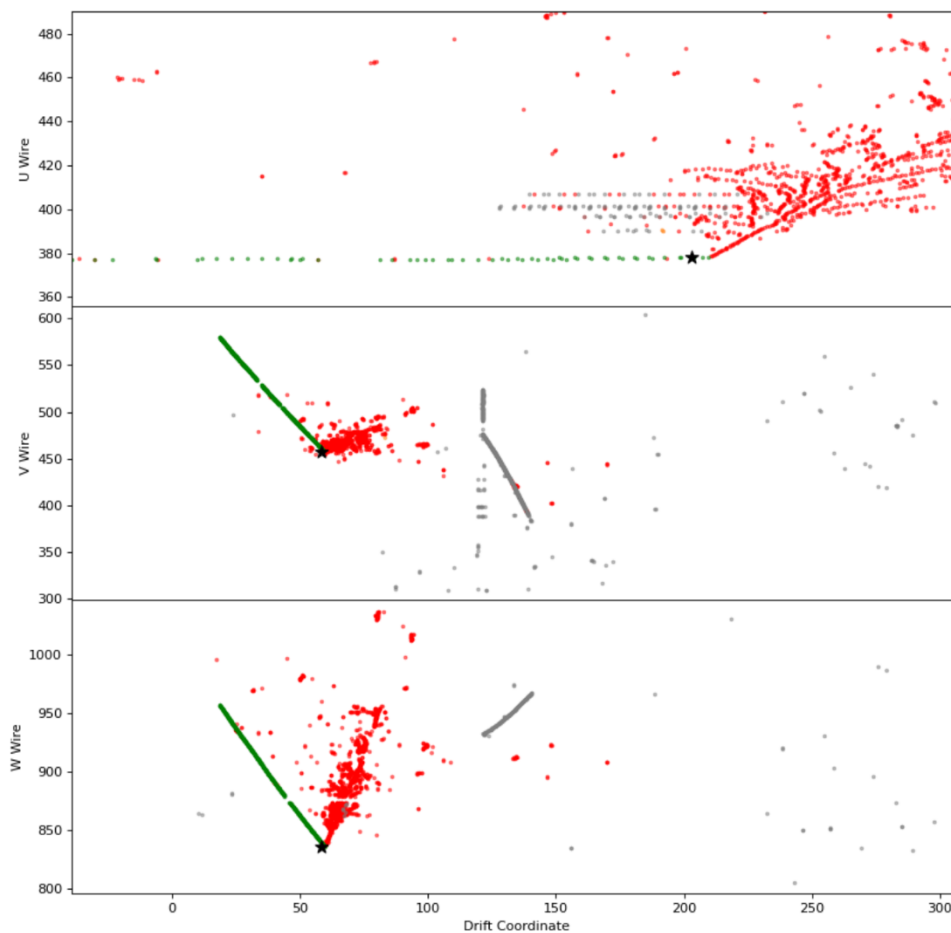
Event Display Options



You will find this panel on the LHS of the image. You can move the image around using option 4 and zoom in using option 5. You can save things by hitting option 6.

9. The next display is of the hits that are in the identified neutrino slice. This event display will help you identify 1) whether we have identified the correct slice 2) if you've lost any of your signal hits 3) if we've brought in a sneaky cosmic.

Again, hits are coloured by their PDG code and uses the same dictionary as above. Cosmics have no truth info, these will be drawn in grey.



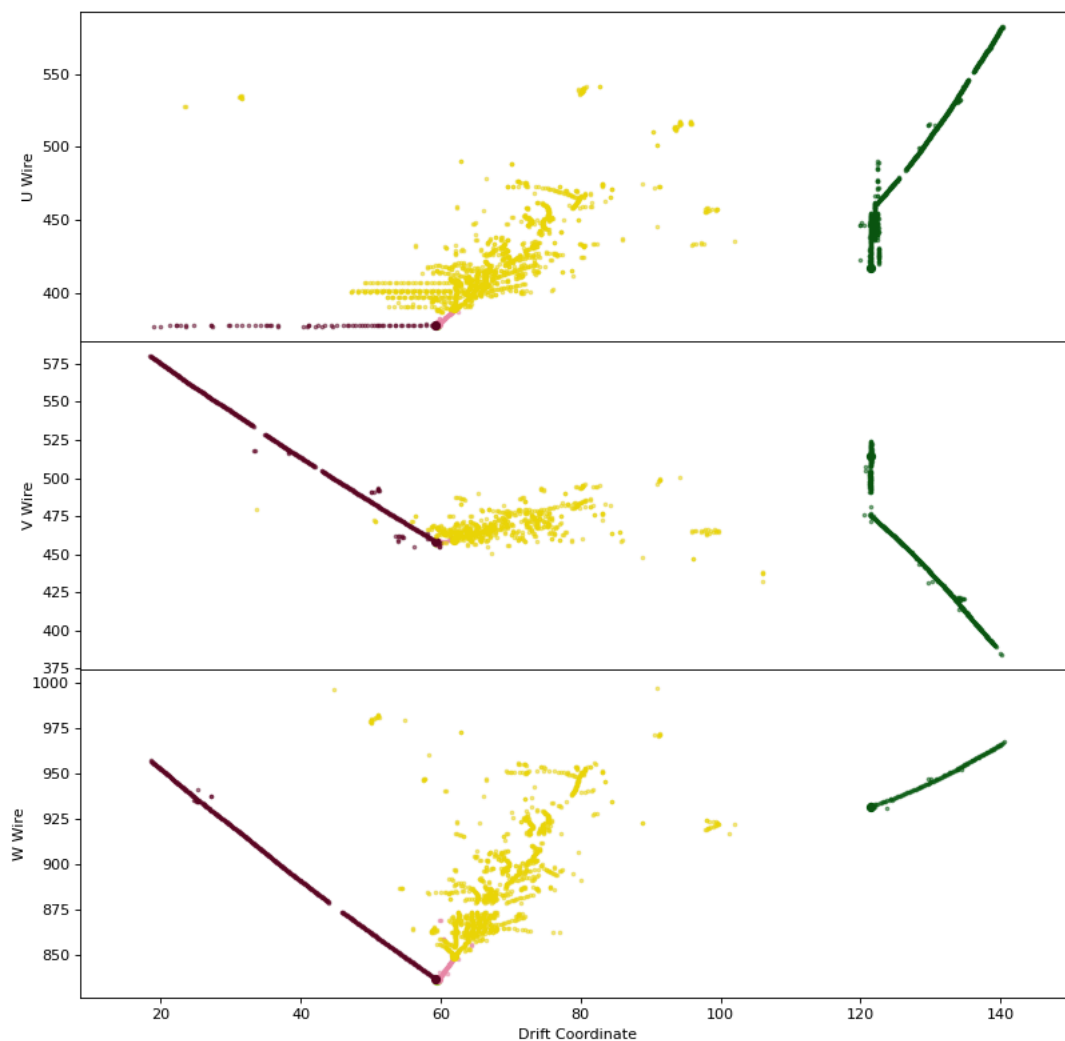
In this example you can see that a cosmic ray has snuck into our neutrino slice. Sneaky sneaky.

Event Display Options

10. Okay, the next panel shows you the pandora reconstructed output. Here the particles can be coloured by 1) by PFP i.e. each PFP has its own colour (this is a random choice, so some annoying colours might appear) 2) their truth matched PDG (if particles are merged together) this might look a bit complicated 3) whether they are track or shower-like 4) their generation i.e. whether they are a child of the neutrino, grandchild etc..

I think the default is by PFP, if you want to change it, find these lines in the code and change the 'colourMode'. You'll have to hit the notebook's forward button again to see the change

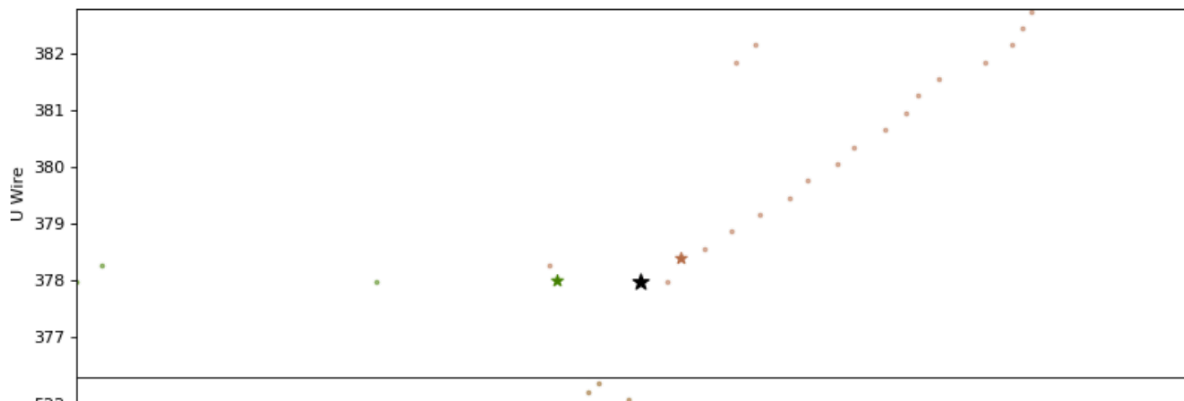
```
#####  
# Get PFP colours  
#####  
  
pdgColours_graph = {13 : 'blue', 11 : 'red', 2212 : 'green', 211 : 'pink', 22 : 'tab:orange'}  
trackShowerColours_graph = {11 : 'blue', 13 : 'red'}  
hierarchyColours_graph = {1 : 'black', 2 : 'red', 3 : 'blue', 4 : 'g'}  
  
# 0 - random, 1 - PDG, 2 - track/shower, 3 - generation  
colourMode = 0  
  
particleColours = []  
  
for iParticle in range(nParticles) :
```



Event Display Options

In this example, I've coloured by PFP. You can see that the electron has been split into into its track-like stub and shower region. Although not in this screenshot (sorry I'm writing this on the train) you'll see stars which show the reco vertices. The reco vertices are 3D objects, and have been projected into each view for visualisation. PFP vertices will be coloured by the colour of their hits. The neutrino vertex will always be black.

So, if I zoomed into the U view, I would see something like this:



11. Okay so the last panel shows the 3D reco output. Again particles are coloured depending on the 'colourMode'. The graph is interactive so you can click and drag and hopefully things should rotate! Again, the stars demonstrate vertices, as coloured by the associated PFP. The neutrino vertex is always black.

