2D Shower Growing and Vertex Finding

Andy Chappell

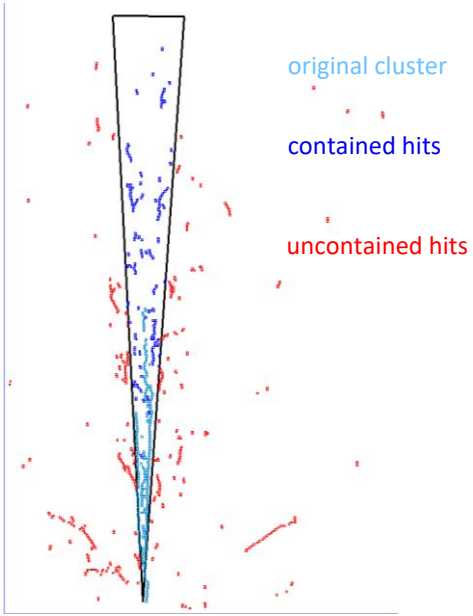
10/05/2021 FD Sim/Reco Meeting

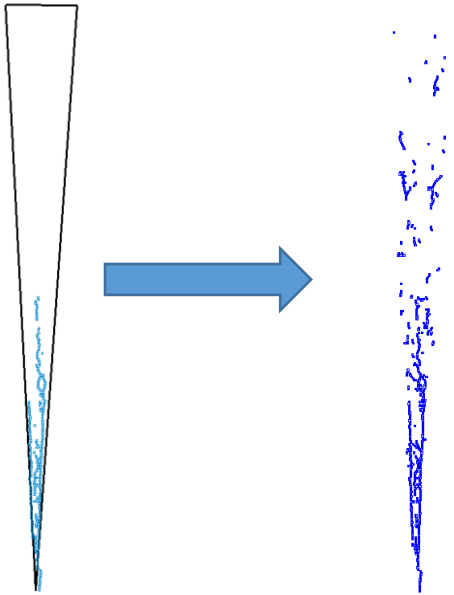
DEEP UNDERGROUND NEUTRINO EXPERIMENT

WARWICK

Previous update

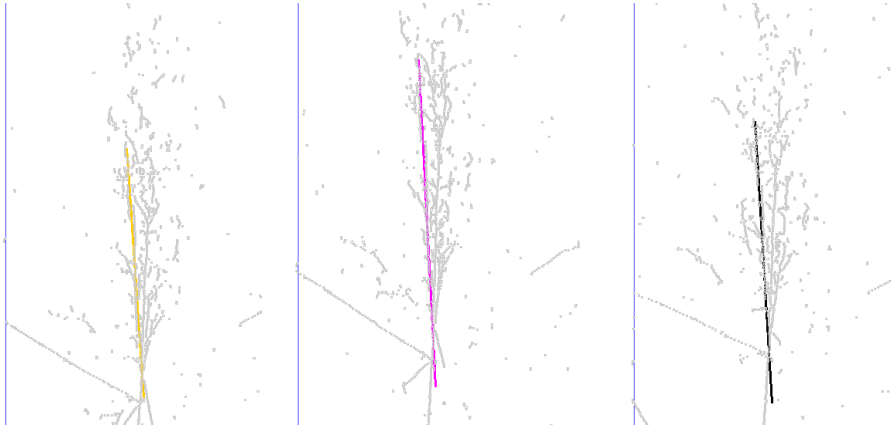
- Check bounds hit intersection
- From this seed…
- We get this

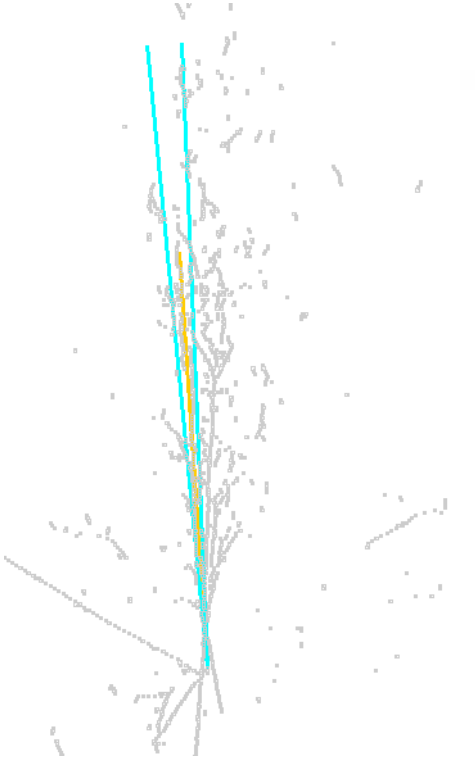




- Given a large enough seed cluster, single view clusters can be grown quite effectively
- Unfortunately, seed clusters are rarely large enough
- Need to allow smaller seed clusters, but PCA becomes less reliable
- Need information from other views…

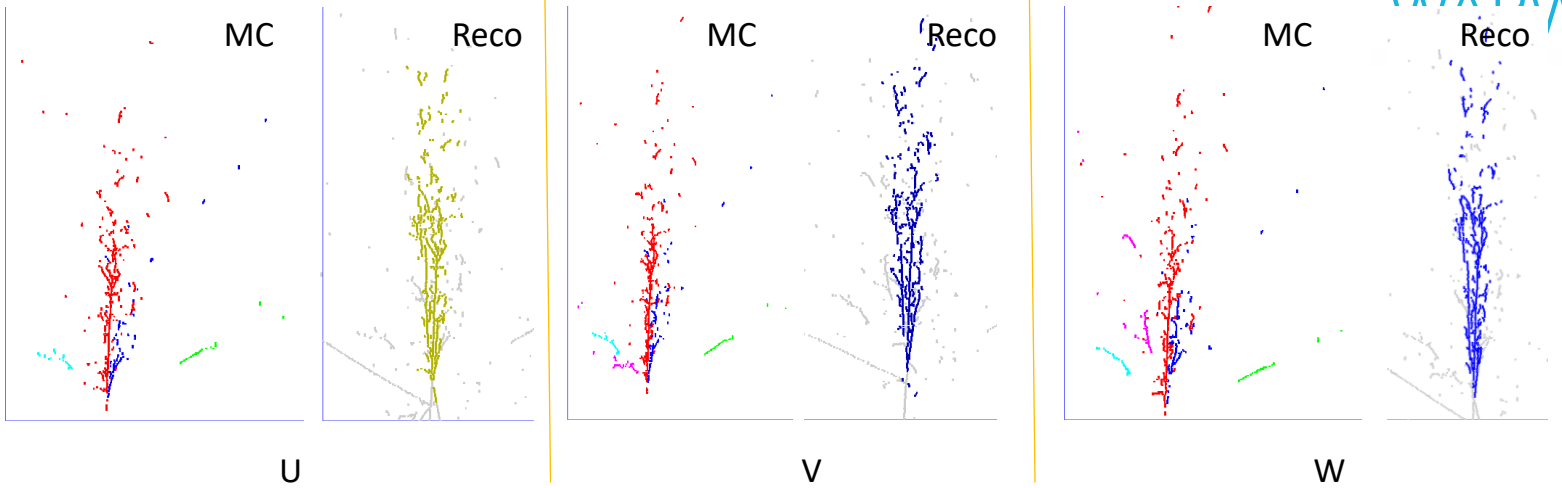
Adding multi-view information





- Example above shows U (left) and W (middle) views used to project PCA axis into the V (right) view
- Aim to alleviate the need for large clusters to form PCA seeds
- Cones built around these axes much like the previous version, but require coherent match across views to merge clusters within each view

Post 2D shower clustering example



- Provisional clustering looks acceptable
  - Ideally three main showers, but given the overlap the result is understandable
- This algorithm does now act in other events
  - Optimisation work is ongoing
- Running grid jobs to assess cluster purity and completeness

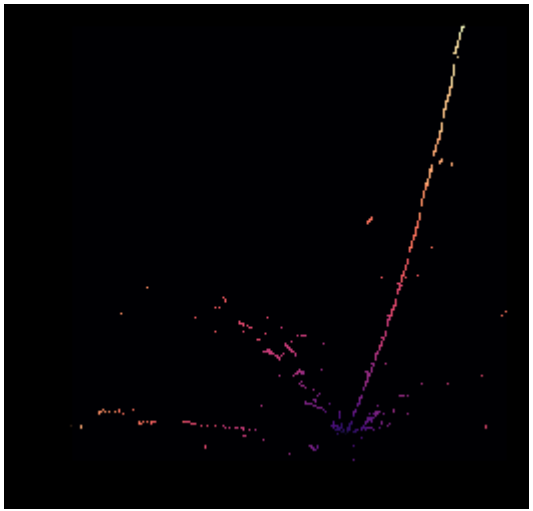
Deep-learned vertexing

WARWICK

- Previous efforts to extract the primary (and secondary) vertices via semantic segmentation have been unsuccessful
  - Likely source of failure is that attempting to tag the vertex leaves the network with very little truth information to work with
- Rethinking how the vertex information is encoded in the truth might overcome this problem, so I've been slowly chipping away at this as a side-project
  - It's not done yet, but I figured there was enough here to provide an update

Encoding truth information

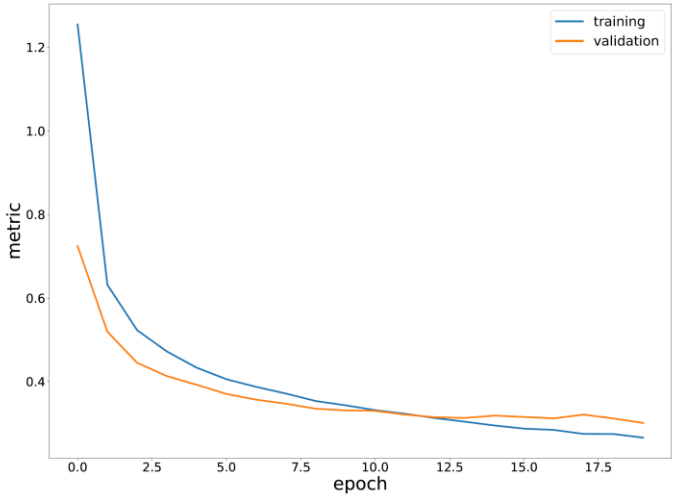
- The distance from each hit to the primary vertex is calculated and assigned to one of 19 classes (the 20th class encodes the null hit)
  - Each class represents a small bin describing a lower and upper distance bound
- Training images therefore look like this:

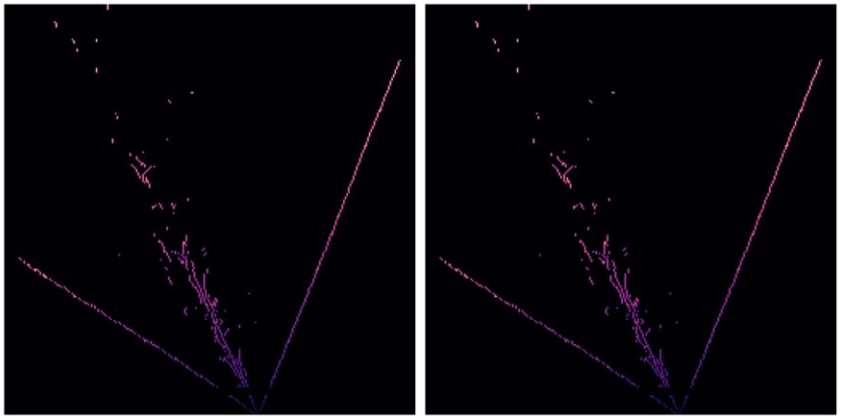


- The distance is a fractional distance relative to the image diagonal in pixels to ensure classes are invariant to the event scale
- This gives the network much more to work with
- Note: I've also modified the colour palette for clarity here, future images will look subtlety different, but convey the same underlying information

Network training

- This truth encoding leads to stable training across all three views, as seen in the loss function evolution below left
- Good correspondence between truth and network classification in the validation set, with a representative example below right



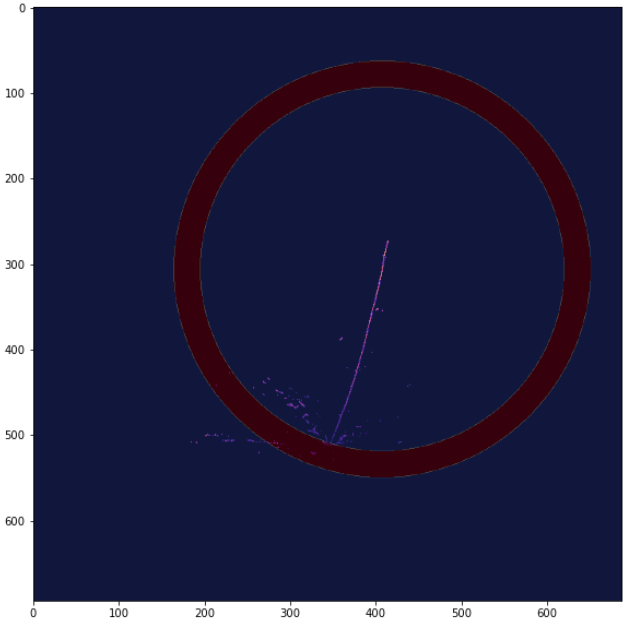


Truth

Network Classification

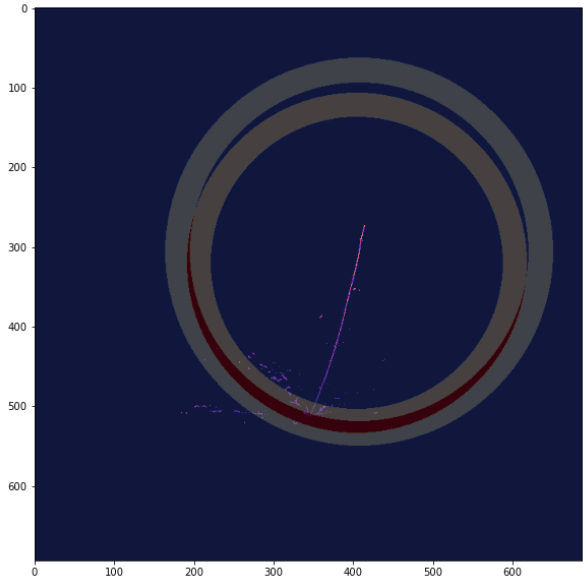
Using the output

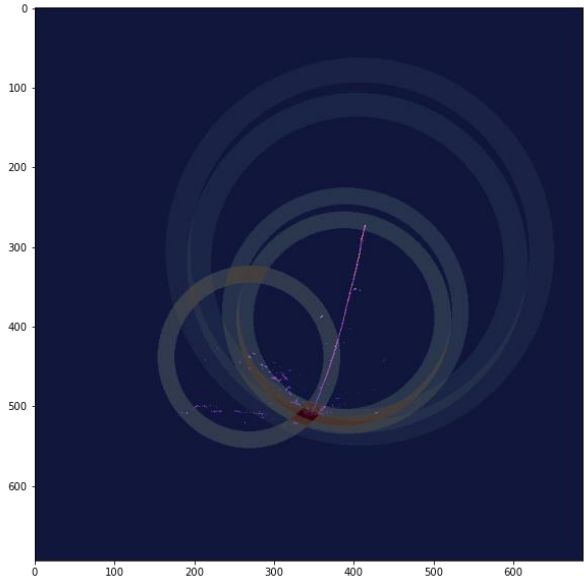
- The network doesn’t return a vertex location in any view
- Need to process the output to determine the likely vertex location
- This is done via consensus
- For each hit, the network classification is converted to the known lower and upper distance bounds and a weighted (inversely proportional to ring area) ring is drawn
- From a single ring, the vertex can be anywhere within the shaded region



Using the output

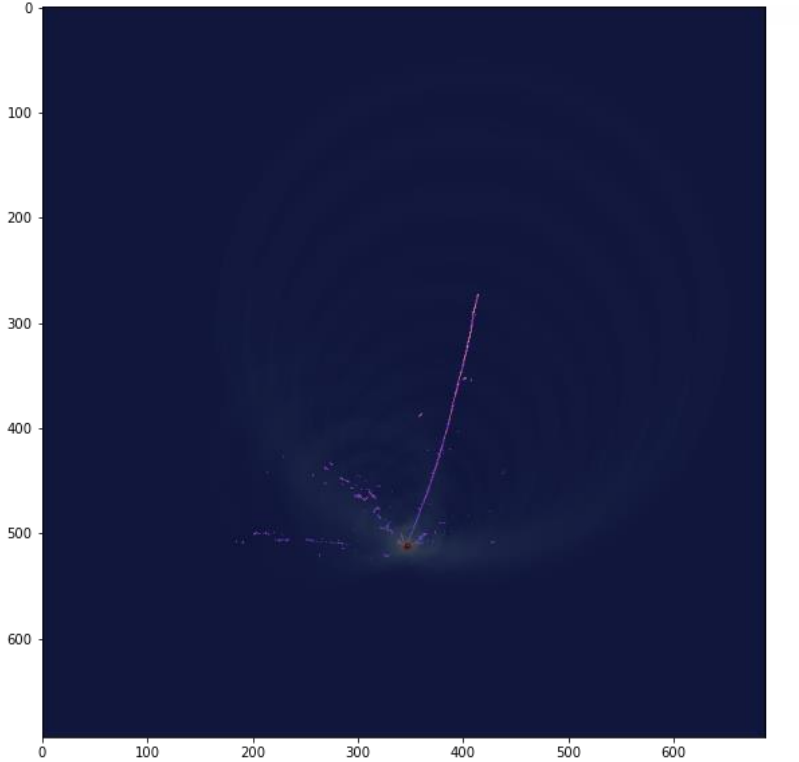
- As we incorporate information from more hits, the overlapping rings isolate the likely vertex location
- This also makes the approach quite robust, errors in some hits get drowned out by the consensus





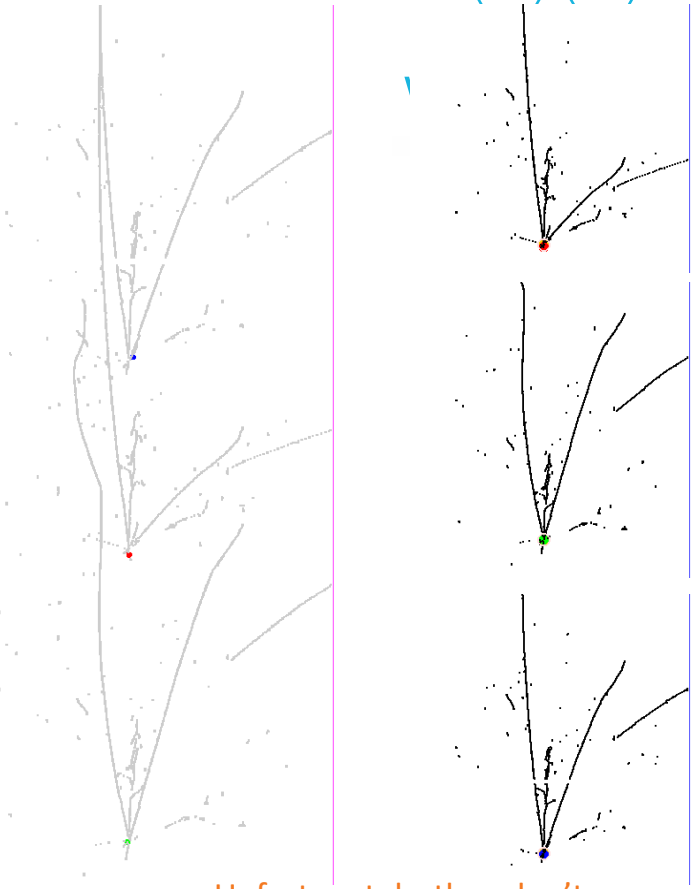
Using the output

- Eventually we get here =>
- With many hits, an efficient ring drawing algorithm is essential
- Bresenham midpoint circle algorithm is a long-standing, efficient circle drawing algorithm using only integer arithmetic
  - Computes filled pixel locations for one octant, then mirrors to the remaining seven octants
- We need rings, not a one-pixel wide circle
- I modified the algorithm to fill between two Bresenham octants, ensuring each interior pixel is filled once, and only once, then mirrored
- All still integer arithmetic, so remains very fast



Running in Pandora

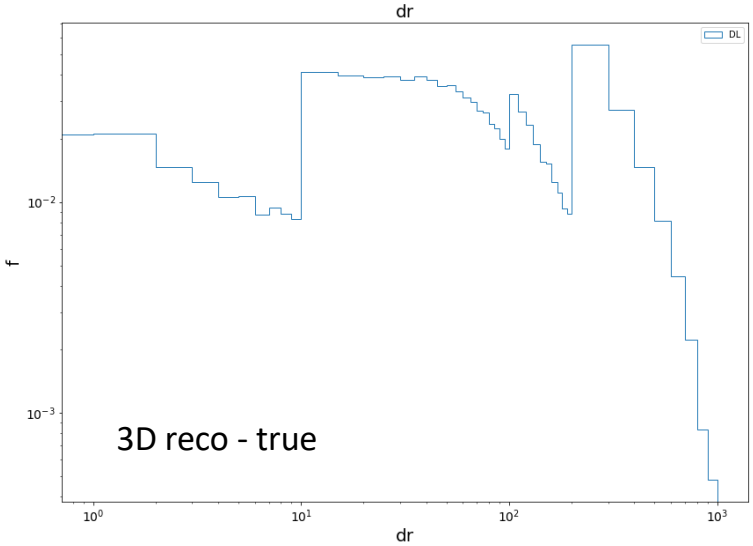
- One nice feature of this network is that it runs on a single image per view, unlike the track/shower network (which routinely runs on 10-20 tiles per view), so the network runs quite quickly
  - < 2 sec/event for complete Pandora pat-rec chain
- Retain scope to run a second, higher resolution pass on the likely vertex region
- Event display on the left indicates the 2D vertex locations extracted in U (red), V (green) and W (blue) views
- Event display on the right indicates the 3D vertex projected into the U, V and W views (top to bottom)

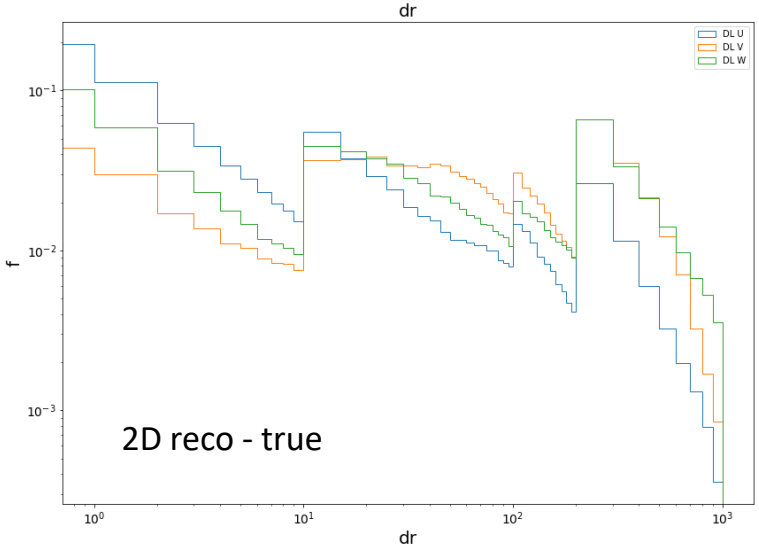


Unfortunately, they don't all look like this yet

Running in Pandora

- There are still issues to resolve
- U view performs notably better than V and W, though not well overall
  - I don't fully understand this yet, but looks like a TorchScript conversion issue (next slide)
  - In Python network classification looks good in all views

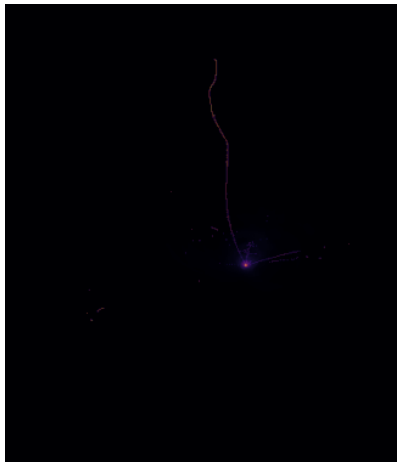




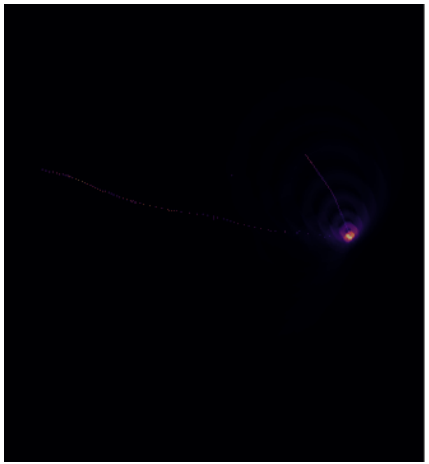
TorchScript v Python

WARWICK

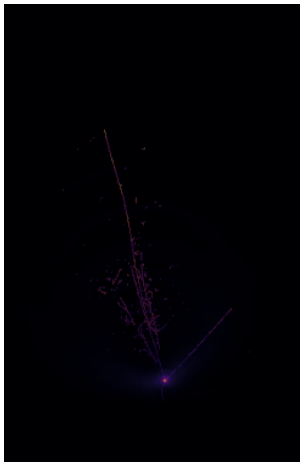
- Selected a small set of events with poor vertex reconstruction in TorchScript context and tested them with the underlying Python networks
- Each image here is from a **different event**
- The networks are clearly working effectively here, so need to understand why these results aren't propagating to the TorchScript context



U



V



W

Next steps

WARWICK

- Try to understand why TorchScript networks aren’t reproducing the original Python performance
- Consider a second higher resolution pass to refine vertex position
- Return multiple candidates from the networks, not just the ‘best’
- Extend the method to secondary vertices