A Shower Reconstruction Algorithm for Electrons

Rory Fitzpatrick, University of Michigan

Tingjun Yang, FNAL

LArSoft Coordination Meeting
September 11, 2018

TCShower

- **Goal:** Efficiently reconstruct electron showers at energies relevant to MicroBooNE and DUNE.
- Accurately identifying a parent electron track means we can use reliable recob::Track information to define the shower vertex, direction, and dE/dx.
- TCShower is a simple but effective algorithm for doing this.

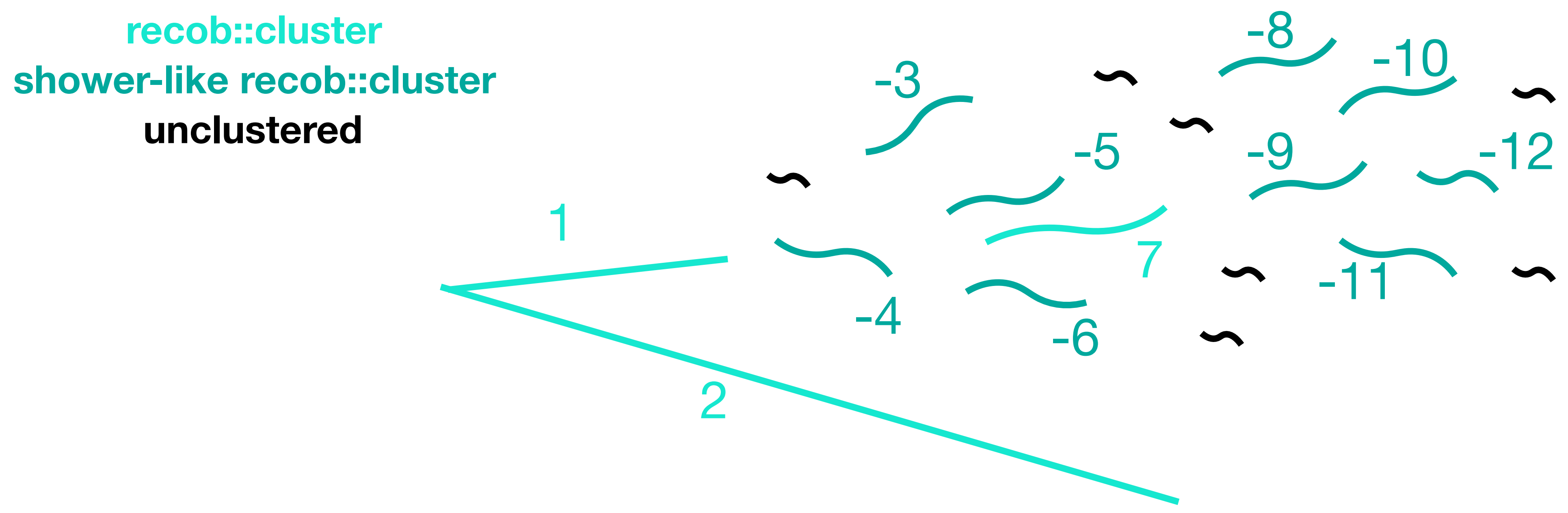
larreco branch: feature/rsf_TCNueSelection

module: larreco/ShowerFinder/TCShower_module.cc
algorithm: larreco/RecoAlg/TCShowerAlg.h(.cxx)
- input: hits, clusters, tracks (plus associations)
- output: showers, hit-shower and shower-slice associations

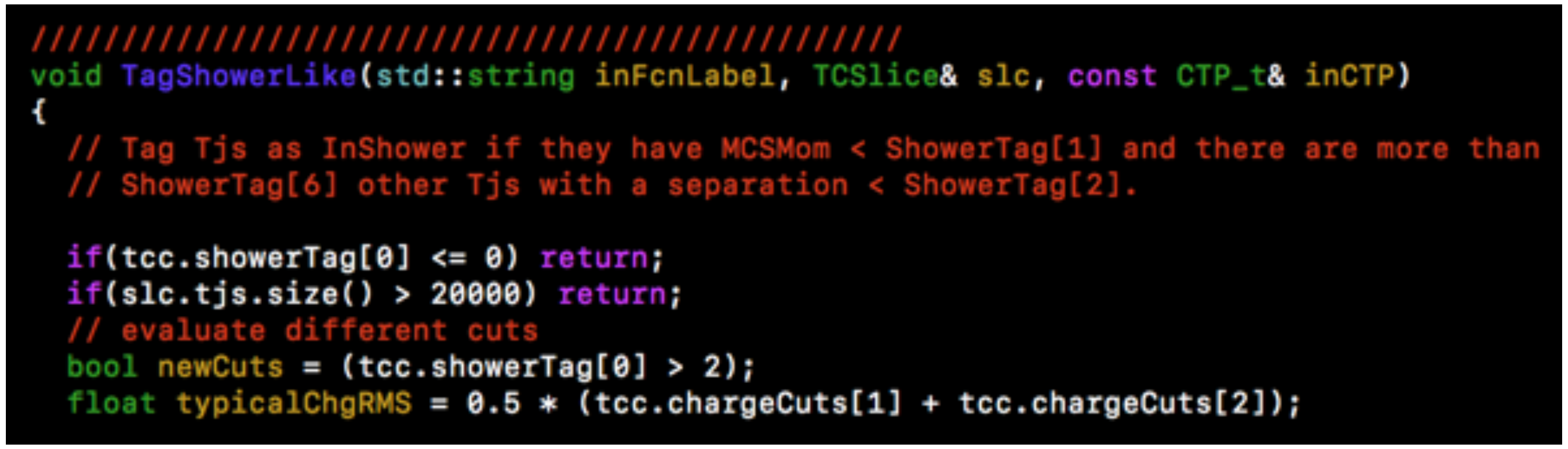
TCShower overview

- "Shower-like" clusters are tagged with negative IDs by trajcluster.
- Iterate through tracks and count how many "shower-like" hits fall within a certain distance of the track axis.
- Simple, adjustable requirements:
    1. minimum and maximum track length
    2. minimum number of shower-like hits
    3. even distribution of shower-like hits around the track
    4. exclude shower-like clusters/hits near vertex of the track
    5. exclude hits "behind" the shower vertex
- If a shower is found, go back and add additional unclustered hits if they are close to the shower axis.
- If a shower is found, look for missing clusters that weren't tagged as shower-like.
- Stop after finding one shower.

1. Tag shower-like clusters



Before using TCShower, run trajcluster and track reconstruction. Trajcluster will label "shower-like" clusters of hits with negative IDs.

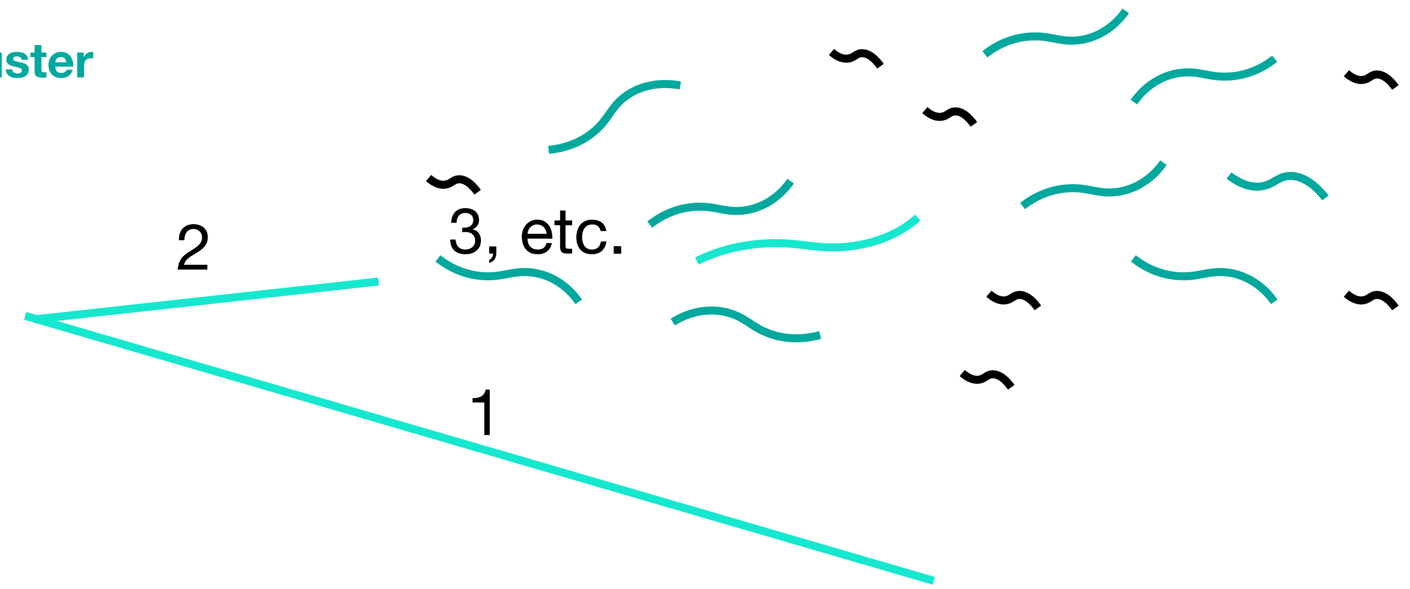


```
////////////////////////////////////////////////
void TagShowerLike(std::string inFcnLabel, TCSlice& slc, const CTP_t& inCTP)
{
  // Tag Tjs as InShower if they have MCSMom < ShowerTag[1] and there are more than
  // ShowerTag[6] other Tjs with a separation < ShowerTag[2].

  if(tcc.showerTag[0] <= 0) return;
  if(slc.tjs.size() > 20000) return;
  // evaluate different cuts
  bool newCuts = (tcc.showerTag[0] > 2);
  float typicalChgRMS = 0.5 * (tcc.chargeCuts[1] + tcc.chargeCuts[2]);
```

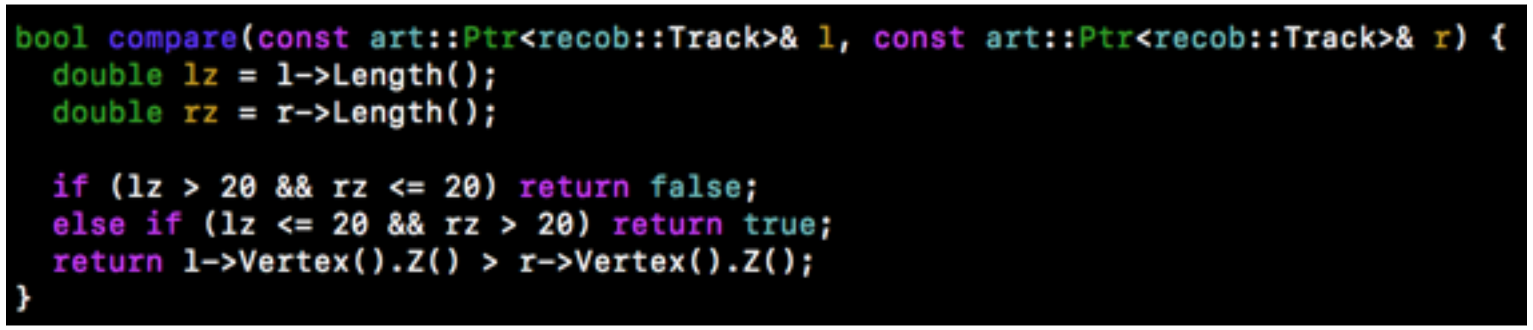
The shower-like decision is made based on MCS momentum and proximity to other clusters with user-defined thresholds (ShowerTag[1] = 100 and ShowerTag[2] = 10 are defaults)

2. Sort Tracks

recob::cluster
shower-like recob::cluster
unclustered



Sort tracks based on length (< 20 cm, > 20 cm) then by start z position. The intention is to test the parent electron (2 in the schematic) prior to tracks reconstructed inside the shower.

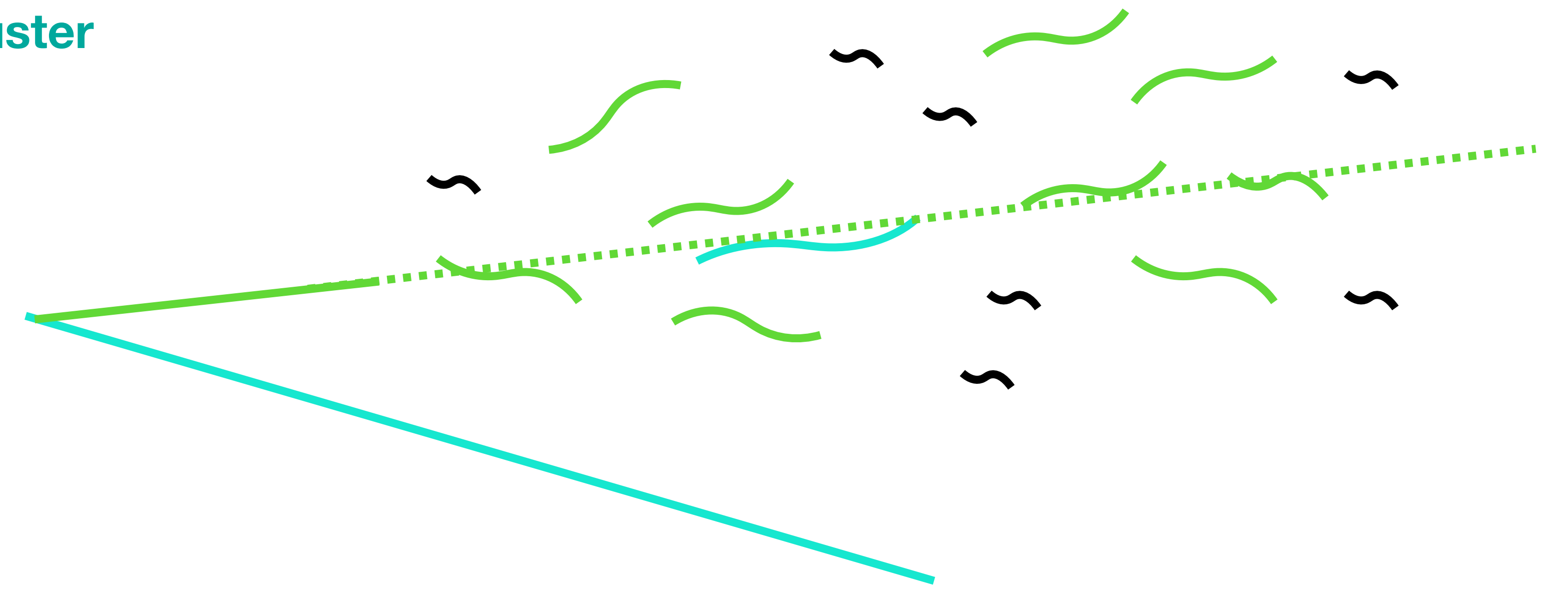


```
bool compare(const art::Ptr<recob::Track>& l, const art::Ptr<recob::Track>& r) {
  double lz = l->Length();
  double rz = r->Length();

  if (lz > 20 && rz <= 20) return false;
  else if (lz <= 20 && rz > 20) return true;
  return l->Vertex().Z() > r->Vertex().Z();
}
```

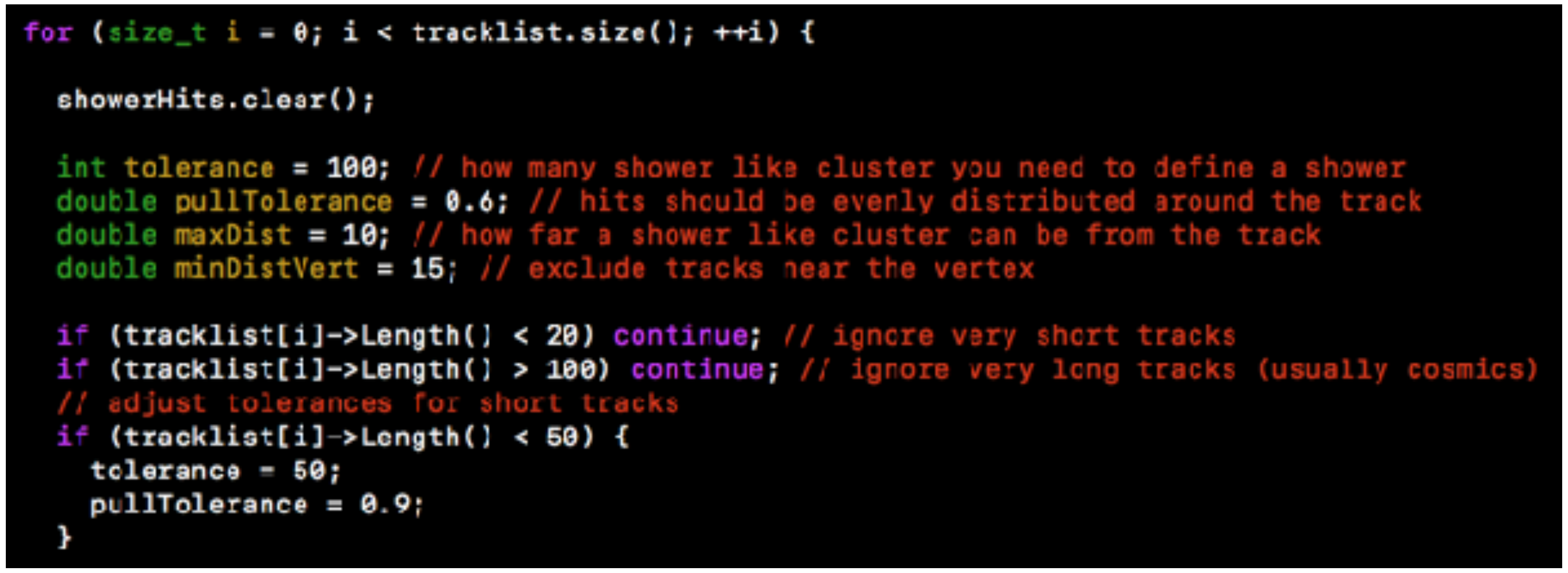
This sorting mechanism works well for DUNE/ArgoNeuT energies because electrons are usually forward-going. Additional sorting mechanisms could be added.

3. Iterate through tracks

recob::cluster
shower-like recob::cluster
unclustered
potential shower



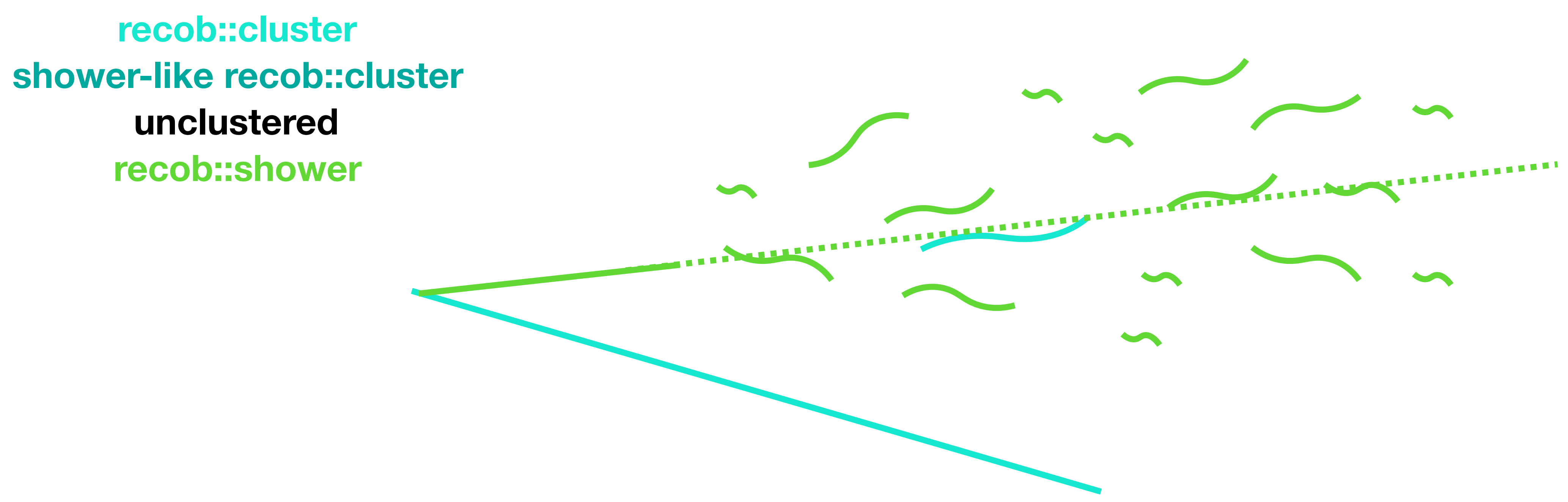
Count "shower-like" hits near the axis defined by the track



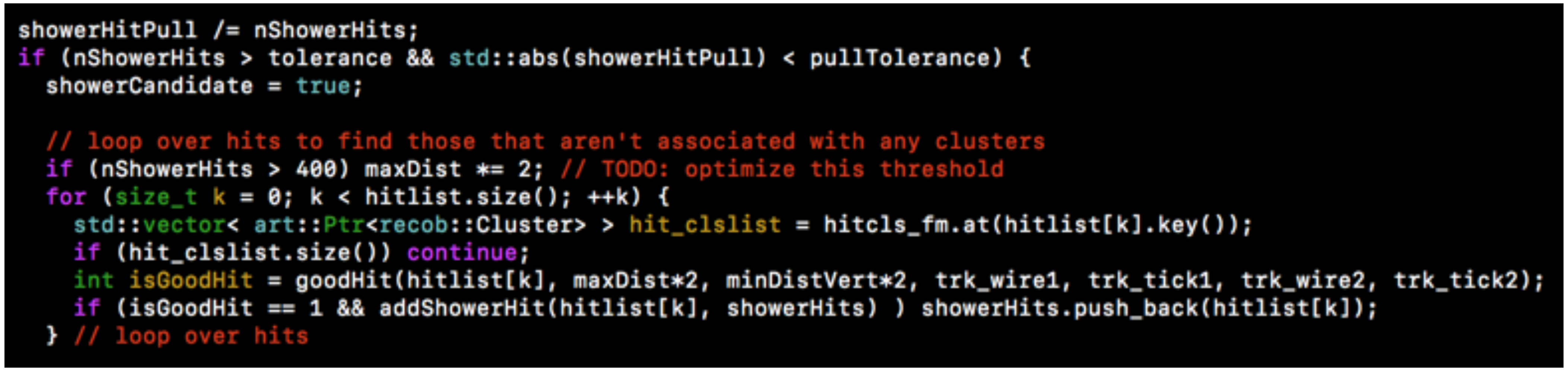
```
for (size_t i = 0; i < tracklist.size(); ++i) {

  showerHits.clear();

  int tolerance = 100; // how many shower like cluster you need to define a shower
  double pullTolerance = 0.6; // hits should be evenly distributed around the track
  double maxDist = 10; // how far a shower like cluster can be from the track
  double minDistVert = 15; // exclude tracks near the vertex

  if (tracklist[i]->Length() < 20) continue; // ignore very short tracks
  if (tracklist[i]->Length() > 100) continue; // ignore very long tracks (usually cosmics)
  // adjust tolerances for short tracks
  if (tracklist[i]->Length() < 50) {
    tolerance = 50;
    pullTolerance = 0.9;
  }
```

Thresholds can be adjusted by the user. These will be turned into fcl parameters soon.

4. If (shower) add missing hits

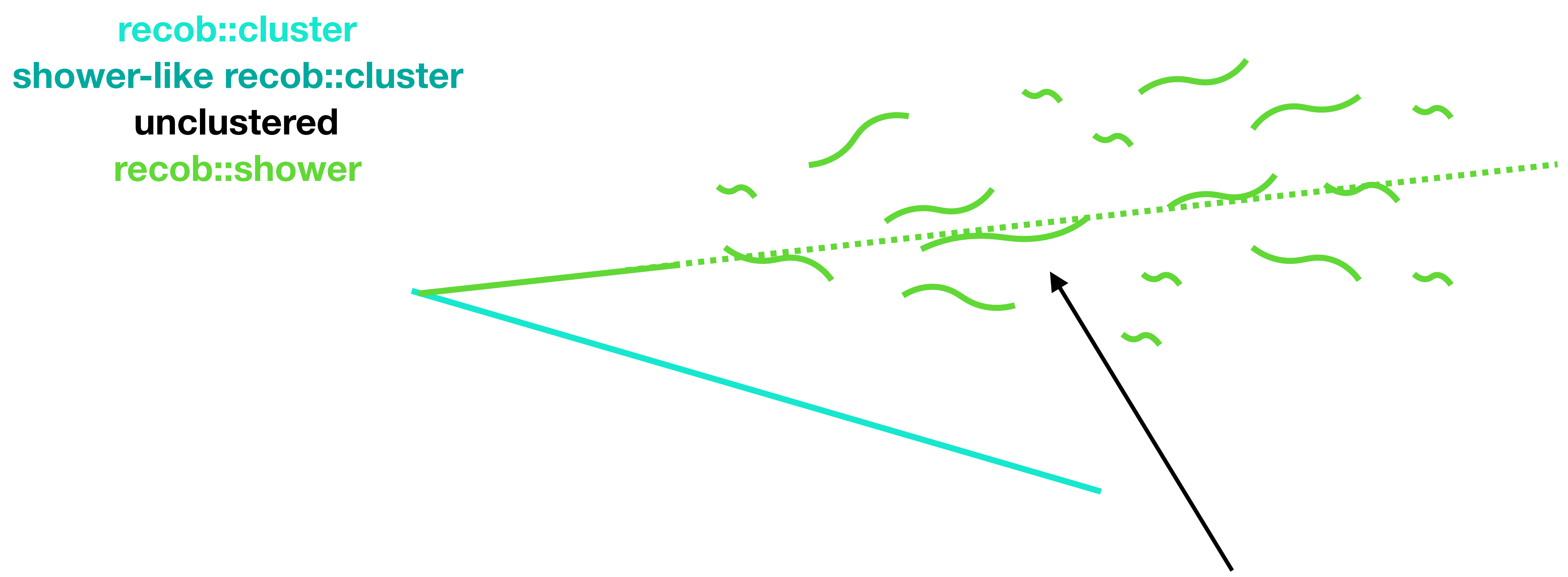


If a shower is found, go back and look for unclustered hits that were ignored before.



```
showerHitPull /= nShowerHits;
if (nShowerHits > tolerance && std::abs(showerHitPull) < pullTolerance) {
  showerCandidate = true;

  // loop over hits to find those that aren't associated with any clusters
  if (nShowerHits > 400) maxDist *= 2; // TODO: optimize this threshold
  for (size_t k = 0; k < hitlist.size(); ++k) {
    std::vector< art::Ptr<recob::Cluster> > hit_clslist = hitcls_fm.at(hitlist[k].key());
    if (hit_clslist.size()) continue;
    int isGoodHit = goodHit(hitlist[k], maxDist*2, minDistVert*2, trk_wire1, trk_tick1, trk_wire2, trk_tick2);
    if (isGoodHit == 1 && addShowerHit(hitlist[k], showerHits) ) showerHits.push_back(hitlist[k]);
  } // loop over hits
```

5. If (shower) add missing clusters

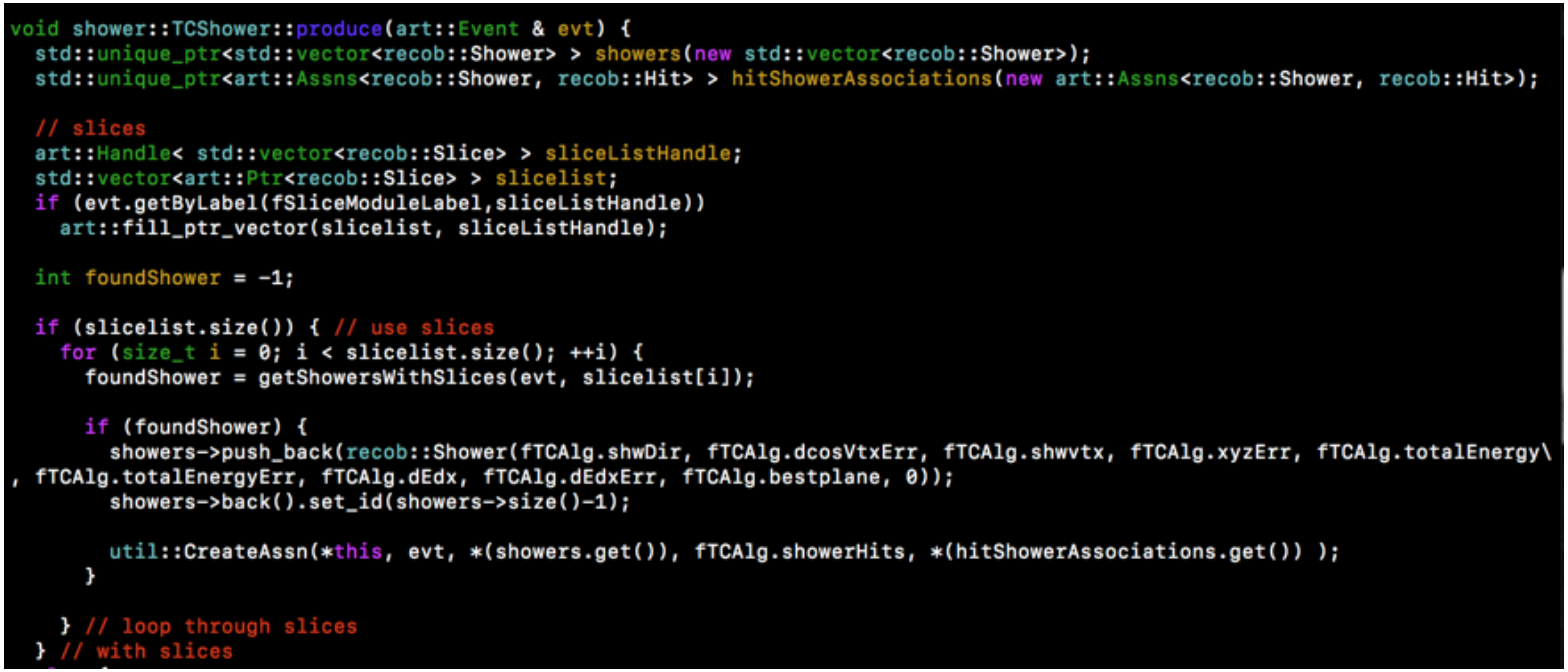


If a shower is found, go back and look for clusters inside the shower that weren't labeled "shower-like".

At this point the shower is complete.

The vertex and direction are defined using track information.
dE/dx can also be extracted from the start of the track.

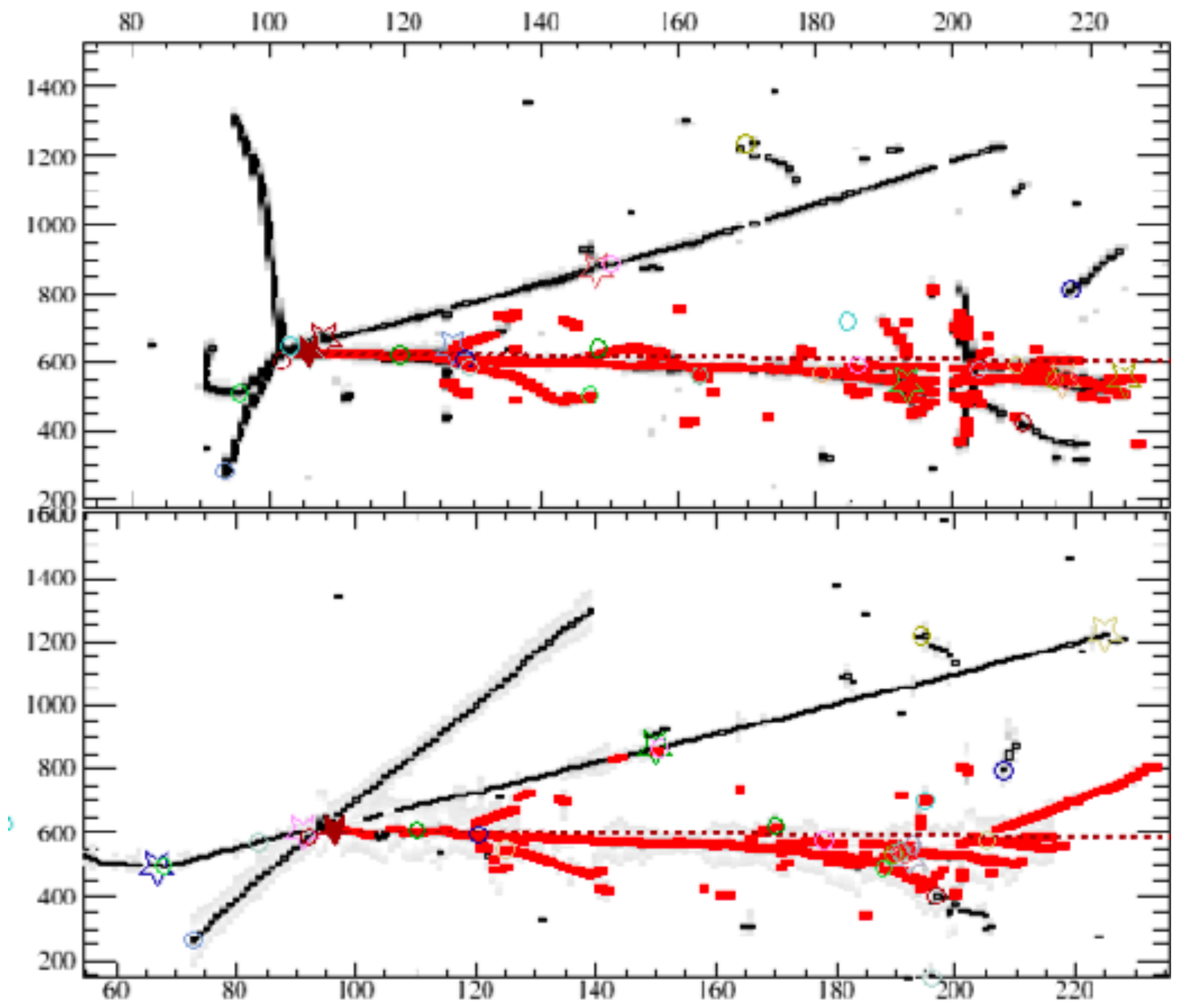
TCShower with recob::slices



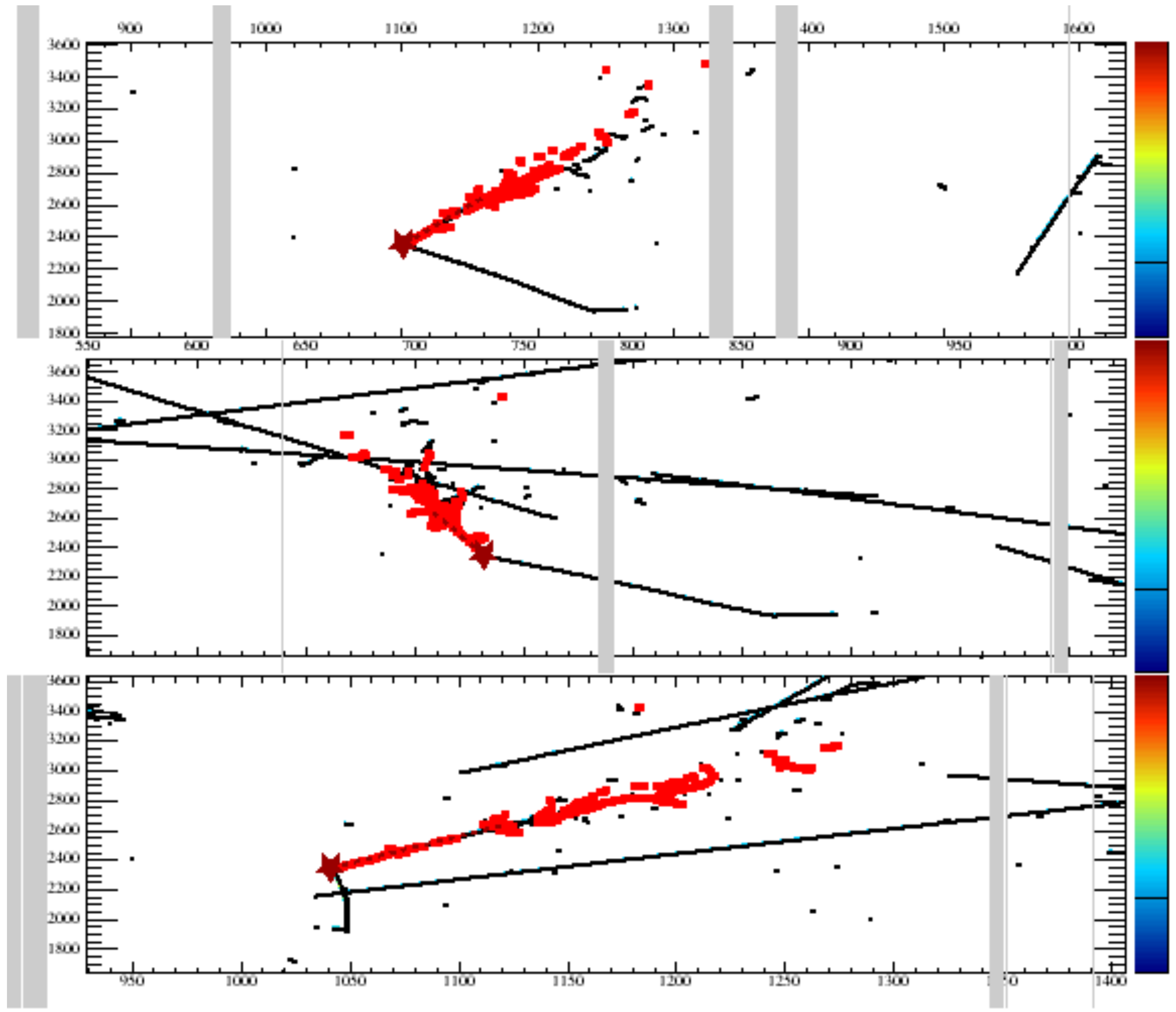
```
void shower::TCShower::produce(art::Event & evt) {
  std::unique_ptr<std::vector<recob::Shower> > showers(new std::vector<recob::Shower>);
  std::unique_ptr<art::Assns<recob::Shower, recob::Hit> > hitShowerAssociations(new art::Assns<recob::Shower, recob::Hit>);

  // slices
  art::Handle< std::vector<recob::Slice> > sliceListHandle;
  std::vector<art::Ptr<recob::Slice> > slicelist;
  if (evt.getByLabel(fSliceModuleLabel,sliceListHandle))
    art::fill_ptr_vector(slicelist, sliceListHandle);

  int foundShower = -1;

  if (slicelist.size()) { // use slices
    for (size_t i = 0; i < slicelist.size(); ++i) {
      foundShower = getShowersWithSlices(evt, slicelist[i]);

      if (foundShower) {
        showers->push_back(recob::Shower(fTCAlg.shwDir, fTCAlg.dcosVtxErr, fTCAlg.shwvtx, fTCAlg.xyzErr, fTCAlg.totalEnergy\
, fTCAlg.totalEnergyErr, fTCAlg.dEdx, fTCAlg.dEdxErr, fTCAlg.bestplane, 0));
        showers->back().set_id(showers->size()-1);

        util::CreateAssn(*this, evt, *(showers.get()), fTCAlg.showerHits, *(hitShowerAssociations.get()) );
      }

    } // loop through slices
  } // with slices
```

Trajcluster was recently restructured to run on recob::slices produced by DBCluster3D. If slices are present, TCShower will run once on each slice.

ArgoNeuT example

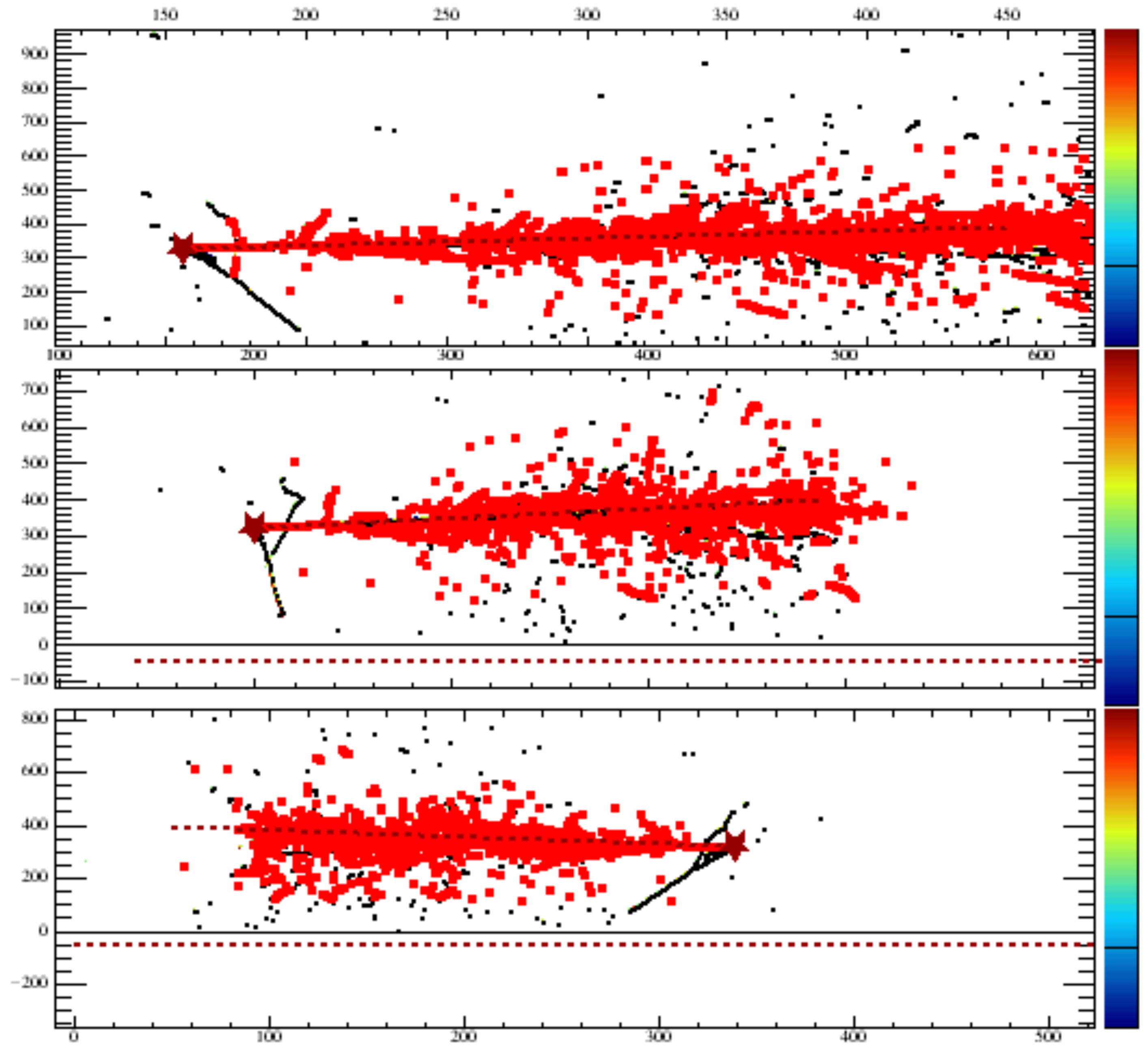


MicroBooNE example



used with recob::slices

DUNE example



TCShower will find showers crossing between TPCs

Summary

- TCShower has been tested successfully on ArgoNeuT, MicroBooNE, and DUNE events.
- Our primary goal is to accurately identify the parent electron to get at the shower vertex, direction, and dE/dx.
- TCShower is designed to find electron showers more efficiently than photon showers (and this is the case in practice).
- *Merging request:* larreco feature/rsf_TCNueSelection