Distinguishing neutron elastic and inelastic scattering through investigation of topological variables

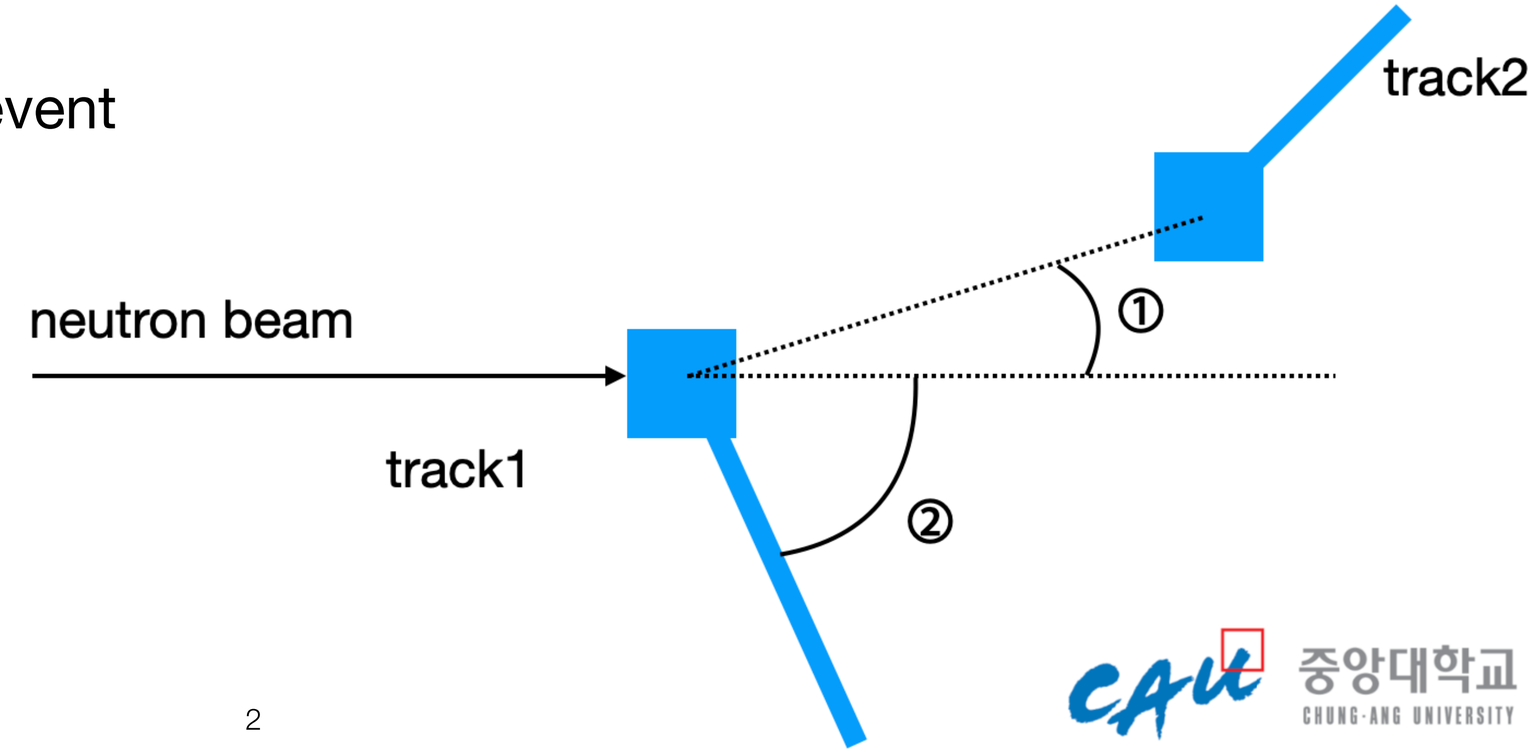
Sunwoo Gwon

Neutron Beam Test Data Analysis Workshop

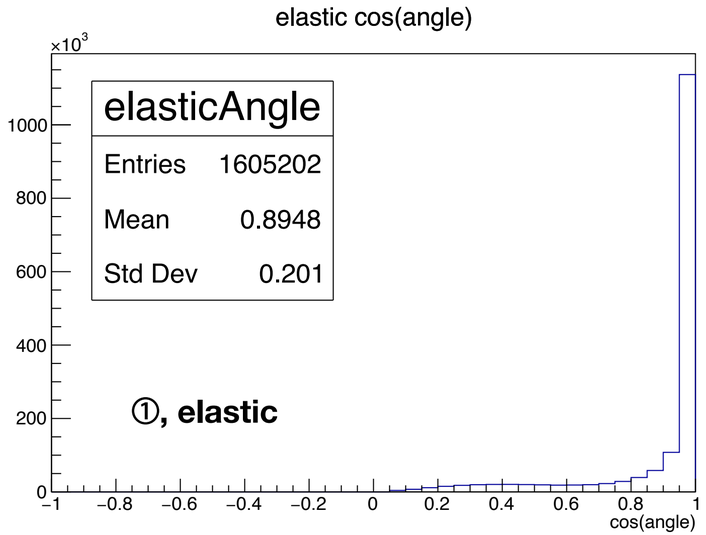
중앙대학교
CHUNG-ANG UNIVERSITY

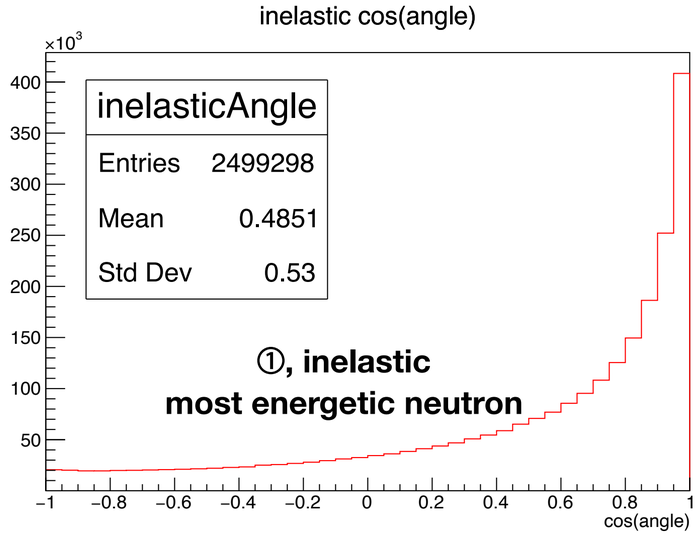
Motivation

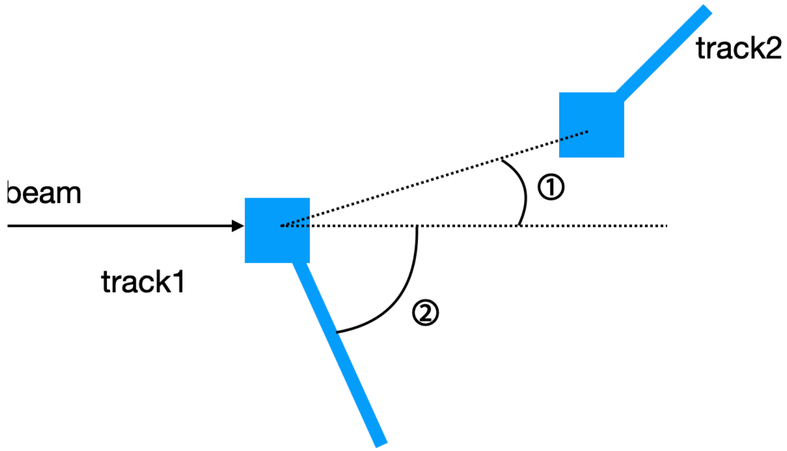
- In GEANT4, neutrons can scatter in two ways; elastic and inelastic.
- We aim to distinguish between elastic and inelastic events using topological variables.
- The ratio between the two events can be utilized as a measure to assess the reliability of the GEANT4 model.
- The target topology is two track event
- The topological variables are neutron scattering angle (①) and first neutron track angle (②).

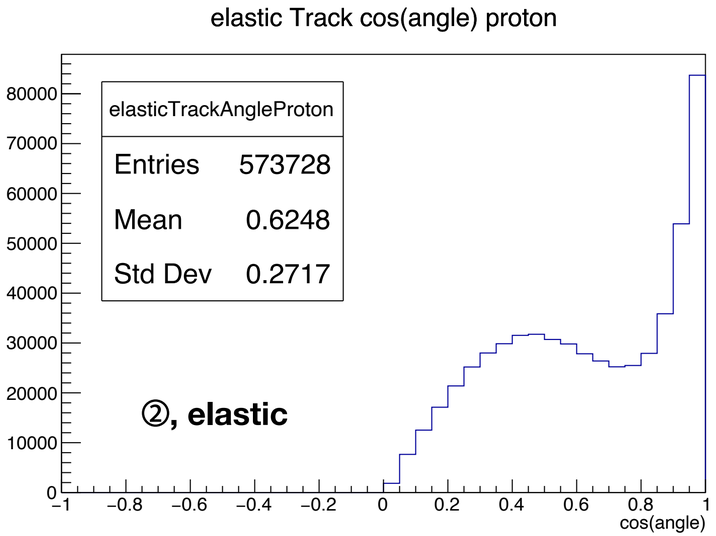


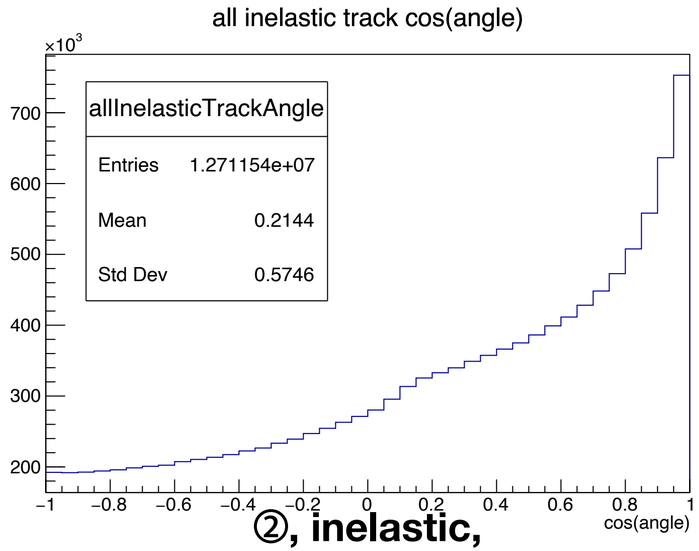
True distribution











- Both of two variables show separation in true level

②, inelastic,
all trajectories (except neutron) induced by the incident neutron

Elastic scattering

- To know whether the scattering is elastic, we can use `GetProcess()` and `GetSubprocess()` in edep-sim `TG4Tracjectory` library.

- `TG4Tracjectory` contains vector of `TG4TrajectoryPoint`, we can access process and subprocess type of each `TG4TrajectoryPoint`.

point (`TG4TrajectoryPoint`)

```
enum G4ProcessType {
    kProcessNotDefined = 0,
    kProcessTransportation = 1,
    kProcessElectromagetic = 2 ,
    kProcessOptical = 3,
    kProcessHadronic = 4,
    kProcessPhotoLeptonHadron = 5,
    kProcessDecay = 6,
    kProcessGeneral = 7,
    kProcessParameterization = 8,
    kProcessUserDefined = 9
};
```

```
// Hadronic subtypes
kSubtypeHadronElastic = 111,
kSubtypeHadronInelastic = 121,
kSubtypeHadronCapture = 131,
kSubtypeHadronChargeExchange = 161,
```

```
point.GetProcess() == 4
&& point.GetSubprocess() == 111
```

→ elastic

Elastic scattering

- Looking at the first hadronic scattering whether it is elastic or not

```
auto trajectroies = event->Trajectories;
auto primaryNeutron = trajectroies.at(0);
```

```
TLorentzVector firstHadronicSccateringPoint;
bool isElastic = false;
for (auto p : primaryNeutron.Points) {
    if (p.GetProcess() == 4) {
        if (p.GetSubprocess() == 121) {
            firstHadronicSccateringPoint = p.GetPosition();
            isElastic = false;
            break;
        } else if (p.GetSubprocess() == 111) {
            firstHadronicSccateringPoint = p.GetPosition();
            isElastic = true;
            break;
        }
    }
}
```

Two track events
preparing hits and clusters

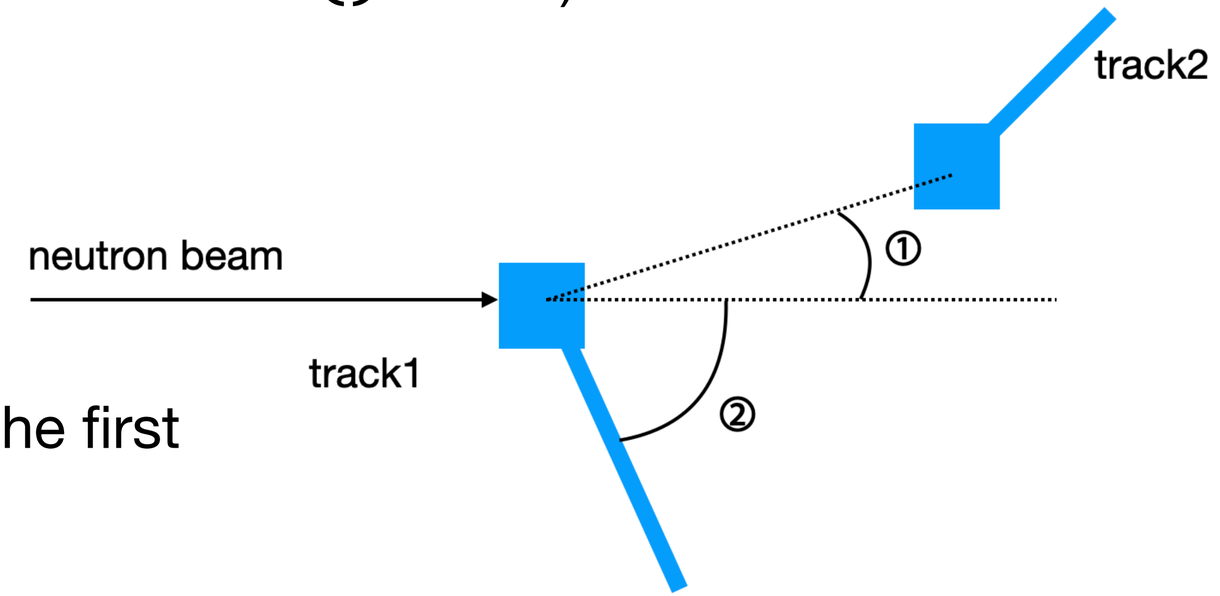
2D hits	3D voxel	clusters
PE cut on hits: 20PE Number of hits: more then three hits `TimeClustering()`	3D view matching of time-clustered hits `MakeHitsToVoxels()`	Voxel clustering (DBSCAN) is used to group voxels into clusters Any voxels within 1.8 cm ($\sqrt{3}$ cm, the diagonal distance between cubes) of each other are grouped into the same cluster

Two track events

- We select two track events (`sfgdCluster.size() == 2`)

```
if (sfgdClusters.size() != 2) continue;

vector<VoxelManager*> tempTrack1 = sfgdClusters[0];
vector<VoxelManager*> tempTrack2 = sfgdClusters[1];
```



- We need to determine which track is the first based on Z

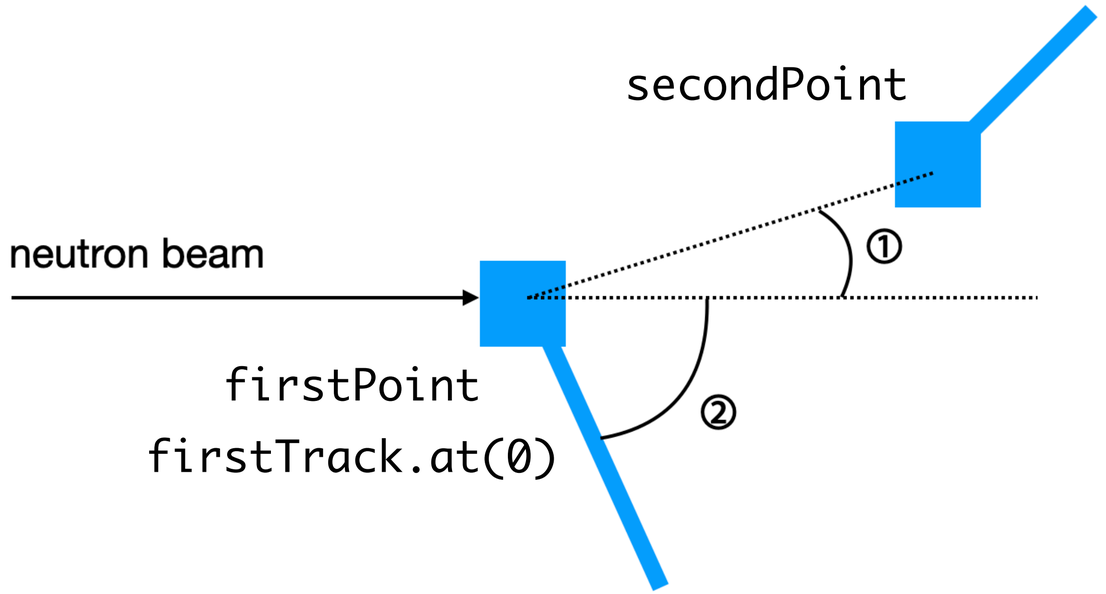
`if (tempTrack1.at(0)->GetZ() < tempTrack2.at(0)->GetZ())`
→ `tempTrack1` is the first track

```
//check which one is the first cluster
if (tempTrack1.at(0)->GetZ() < tempTrack2.at(0)->GetZ()) {
    firstTrack = tempTrack1;
    secondTrack = tempTrack2;
} else {
    firstTrack = tempTrack2;
    secondTrack = tempTrack1;
}
```

Two track events

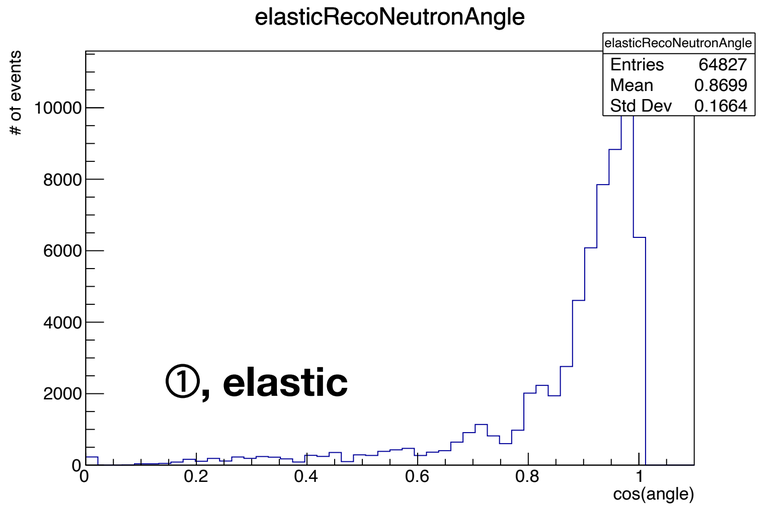
- Two topological variables
  neutron scattering angle (①)
  and first neutron track angle (②).

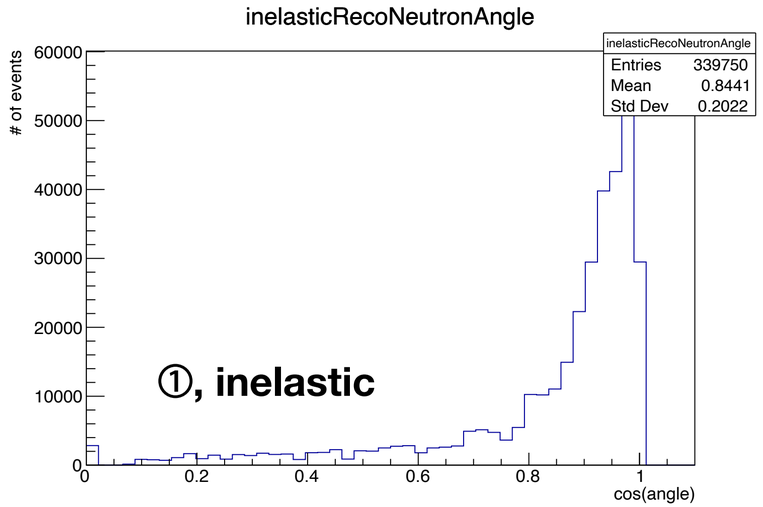
- ① : angle between two vectors
  neutron beam (0, 0, 1) and
  `secondPoint - firstPoint`

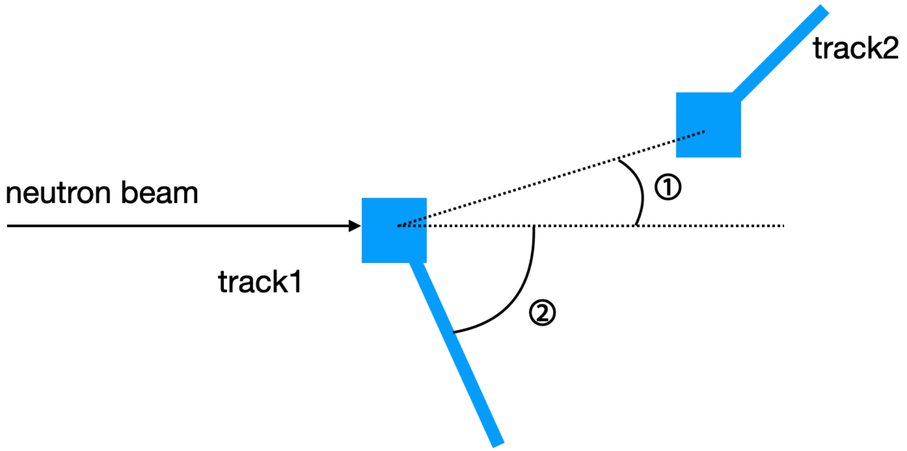


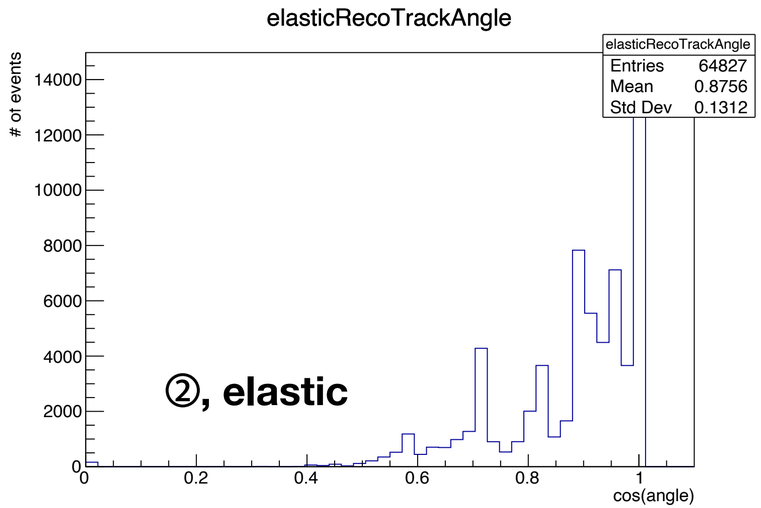
- ② : angle between two vectors
  neutron beam and
  `firstTrack.at(firstTrack.size() - 1) - firstTrack.at(0)`

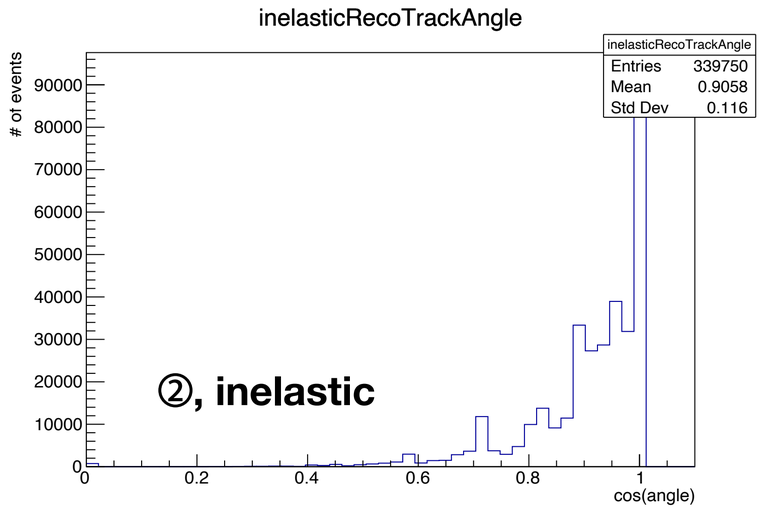
Two track events









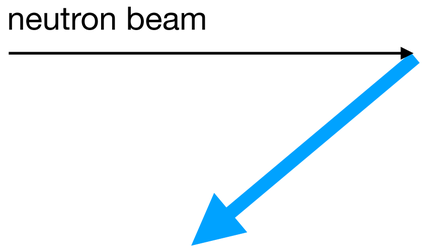


- The current algorithm doesn't have a backward going neutron or track
- Regardless backward going, it's still weird.

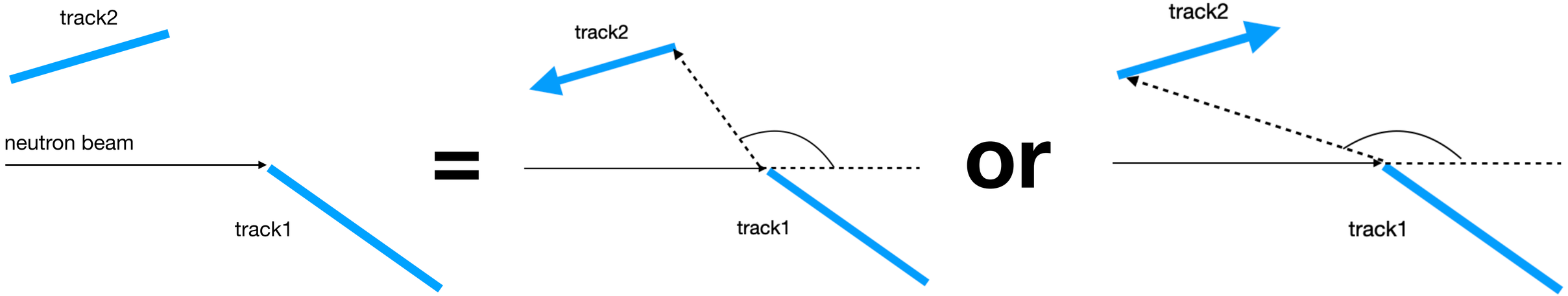
중앙대학교 CHUNG-ANG UNIVERSITY

Possible Improvement

- Implement backward going?

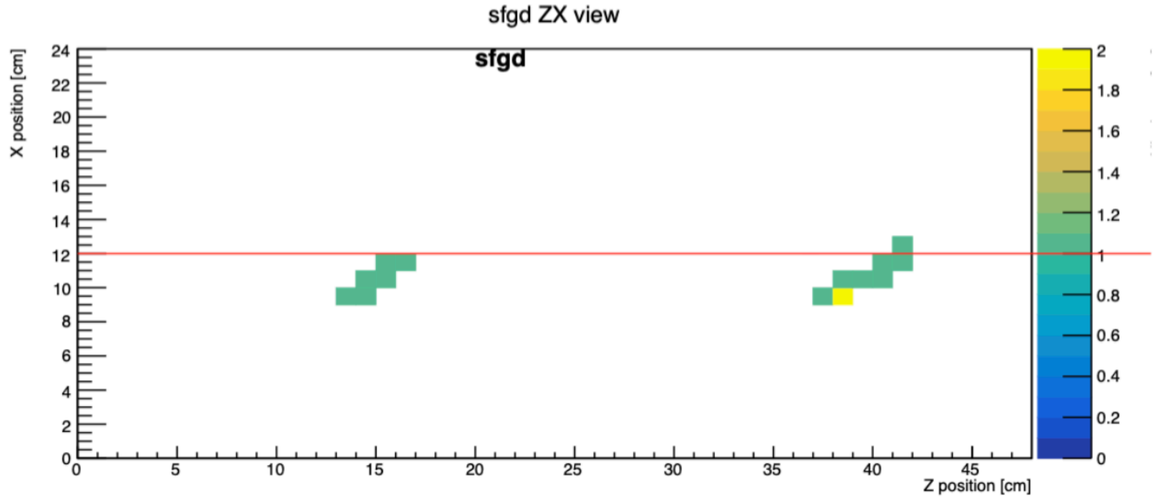


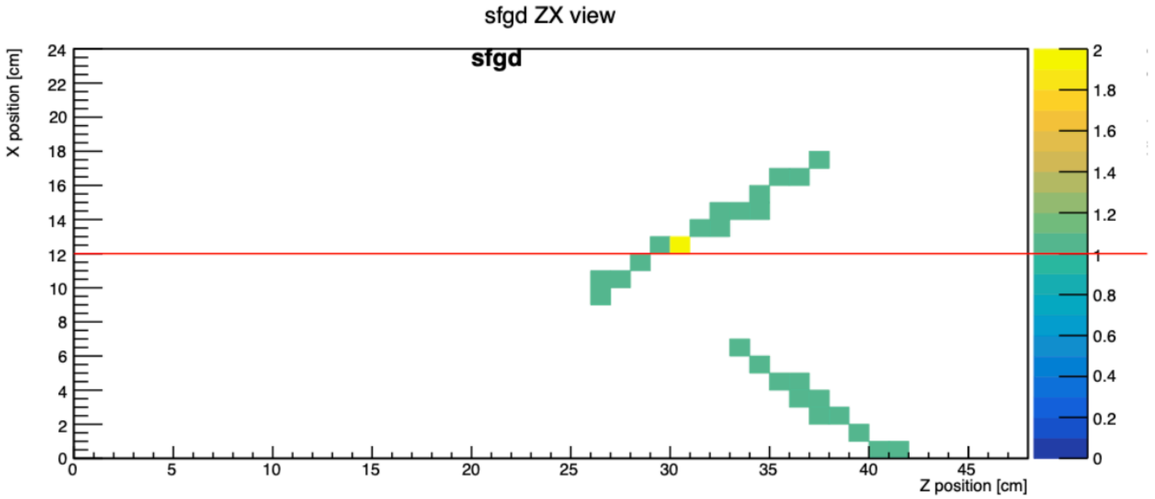
backward going track:
spread decreases as `voxel->GetZ()` increase?

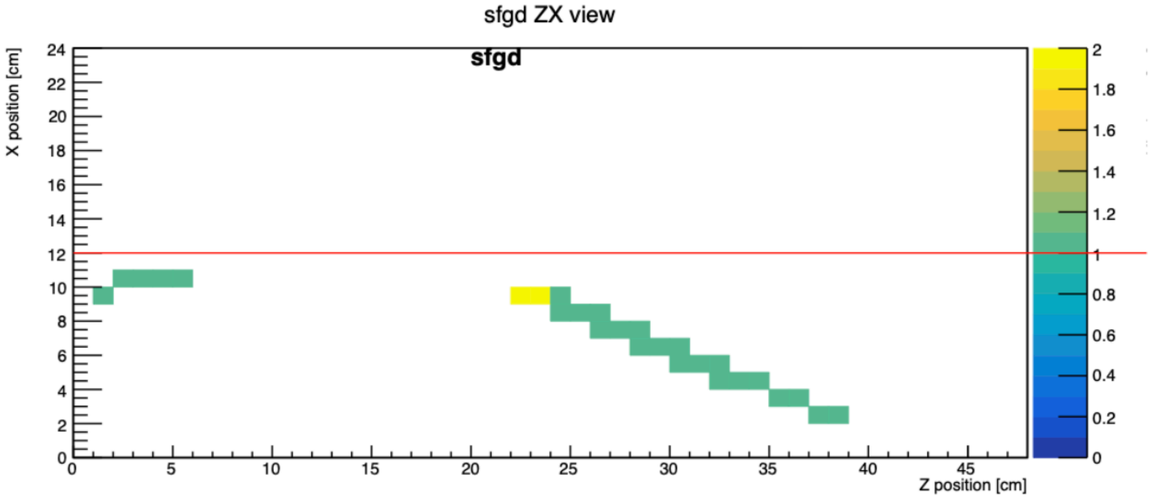


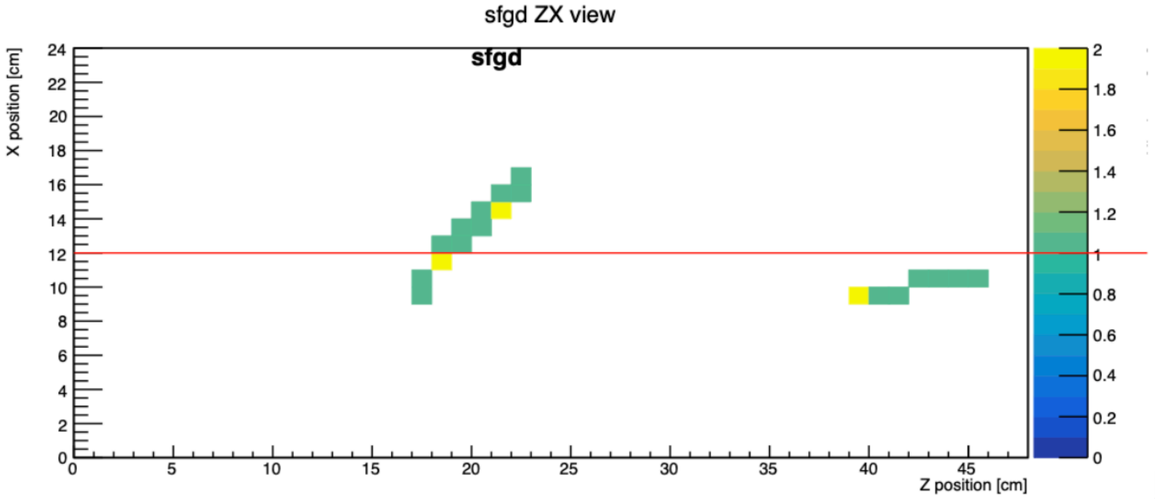
Back up

Two track events









Two track events
preparing hits and clusters

```
int SFGDN = SFGDunpackEvent->GetNHits();

vector<Hit*> sfgdHits = {};
for (int i = 0; i < SFGDN; ++i) {
    Hit* tempHit = (Hit*)SFGDunpackEvent->GetHits()->At(i);

    if (tempHit->GetPE() > 20) {    //threshold = 20pe
        sfgdHits.push_back(tempHit);
    }
}

//Skip event if there are less than 3 hits in the event
if (sfgdHits.size() < 3) continue;

std::vector<vector<Hit*>> TChits = TimeClustering(sfgdHits);

vector<VoxelManager*> sfgdVoxels = MakeHitsToVoxels(TChits[0], 1);
std::vector<vector<VoxelManager*>> sfgdClusters = FindClusters(sfgdVoxels, 1, 3, 1.8);
```

applying threshold

DB clustering voxels (1.8cm)