



# The BackTracker

Brian Rebel  
April, 2014



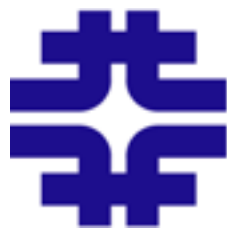
# Back Tracking



- Need a way to map reconstructed objects to the Monte Carlo truth information
- Allows for the evaluation of the reconstruction through several levels of mapping
  - Cell to total collection of `sim::FLSHits` or `sim::PhotonSignals`
  - Hit to collection of `cheat::TrackIDE` (track id and energy) structs
  - Hit to collection of `sim::Particles`
  - Hit to weighted XYZ position of all particles passing through it
  - Collection of `rb::CellHits` to `sim::Particles` contributing to them
  - Collection of `rb::CellHits` to `cheat::TrackIDE` structs
  - Energy deposited by a given particle in a `rb::CellHit`
- Can also determine the purity and efficiency of a collection of `rb::CellHits` given a set of track ids to check against



# Back Tracking



- Provides a convenient way to also figure out truth to truth mappings
  - G4 Track ID to simb::MCTruth object
  - G4 Track ID to simb::Mother particle
  - sim::Particle to simb::MCTruth object
  - simb::MCTruth object to all sim::Particles
  - Collection of rb::CellHits to sim::Particles contributing to them
  - Collection of rb::CellHits to cheat::TrackIDE structs
- Provides a direct link to the sim::ParticleNavigator as well - most of the above methods use the ParticleNavigator in some way or another
- Many other mappings available - look at the MCCheater/BackTracker.h to see what is available



# How to Use the BackTracker



- The BackTracker is a service, so you need to be sure it is defined in the user services block of your .fcl file
- Also include the .h file in your `_module.cc` file or `.cxx` file, i.e.  

```
#include "MCCheater/BackTracker.h"
```
- Then, in your code, grab the service handle by doing  

```
art::ServiceHandle<cheat::BackTracker> bt;
```
- Next decide what you want to learn from the BackTracker



# Grabbing the Particles in the Event



- Maybe you just want to see what particles are in the event

```
sim::ParticleNavigator const& pn = bt->ParticleNavigator();
```
- The ParticleNavigator behaves a lot like a map, has ability to provide iterators over the collection of particles
- Then use the navigator to loop over the sim::Particles in the event

```
for(auto itr = pn->begin(); itr != pn->end(); ++itr){  
    const sim::Particle* part = (*itr).second;  
  
    // do something here with the sim::Particle  
}
```



# Figure out which Particle contributed the most light to a Hit



- Take a `rb::CellHit` get the `sim::Particle` that contributed the most light to make it

```
const sim::Particle* part = bt->HitToParticle(rb::CellHit);
```

- Can do the same thing for a collection of hits from a cluster, prong, etc

```
const std::vector<const sim::Particle*> parts = bt->HitsToParticles(hits);
```

- Use the functions to determine if your hit collection corresponds to the particles you are interested in or not



# Checking Purity and Efficiency



- One way to evaluate the quality of reconstruction is to determine how pure and efficient the algorithm is
- BackTracker has functions to tell you the purity and efficiency of a collection of hits for a given set of track IDs
- Can return maps of track ID to purity/efficiency
- Simply use the `BackTracker::HitCollectionEfficiency`, `BackTracker::HitCollectionPurity` methods