ParticleID and other MicroBooNE MCC9 Updates

Larsoft Coordination Meeting
Nov. 2, 20211

H. Greenlee

Outline

- Historical overview
- ParticleID update
- Other MCC9 updates
- Requests and proposals

Historical Overview

- Original presentation for proposed anab::ParticleID update (breaking change) was made by Adam Lister and Kirsty Duffy in July 17, 2018 larsoft coordination meeting (link to agenda).
- Second presentaiton on anab::ParticleID update was made in Jan. 15, 2019 larsoft coordination meeting (link to agenda).
  - Agreed to in principle, pending implementation of ioread rule.
- MicroBooNE forked MCC9 off of larsoft v08_05_00 (Jan. 16, 2021).
  - ParticleID was eventually merged in MCC9 fork.
  - ParticleID was never merged in develop branch or integration release because of lack of ioread rule … until now.

Old anab::ParticleID Class

- The old (integration release) anab::ParticleID class is hard-wired to store values from a fixed set of particle id algorithms.

```
class ParticleID{
public:

  ParticleID();

  int    fPdg;             ///< determined particle ID
  int    fNdf;             ///< ndf for chi2 test
  double fMinChi2;         ///< Minimum reduced chi2
  double fDeltaChi2;       ///< difference between two lowest reduced chi2's
  double fChi2Proton;      ///< reduced chi2 using proton template
  double fChi2Kaon;        ///< reduced chi2 using kaon template
  double fChi2Pion;        ///< reduced chi2 using pion template
  double fChi2Muon;        ///< reduced chi2 using muon template
  double fMissingE;        ///< missing energy from dead wires for contained particle
  double fMissingEavg;     ///< missing energy from dead wires using average dEdx
  double fPIDA;            ///< PID developed by Bruce Baller
  geo::PlaneID fPlaneID;
```

New anab::ParticleID Class

- The proposed new (and MCC9) anab::ParticleID class holds an arbitrary size collection of algorithm structs.

```
class ParticleID{
public:

  ParticleID();

  std::vector<sParticleIDAlgScores> fParticleIDAlgScores;
                ///< Vector of structs to hold outputs from generic PID algorithms
```

Particle ID Algorithm Struct

```
struct sParticleIDAlgScores { ///< determined particle ID
  std::string fAlgName;
     ///< Algorithm name (to be defined by experiment). Set to "AlgNameNotSet" by default.
  kVariableType fVariableType;
     ///< Variable type enum: defined in ParticleID_VariableTypeEnums.h. Set to kNotSet by default.
  kTrackDir fTrackDir;
     ///< Track direction enum: defined in ParticleID_VariableTypeEnums.h. Set to kNoDirection by
          default.
  int fNdf;
     ///< Number of degrees of freedom used by algorithm, if applicable. Set to -9999 by default.
  int fAssumedPdg;
     ///< PDG of particle hypothesis assumed by algorithm, if applicable. Set to 0 by default.
  float fValue; ///< Result of Particle ID algorithm/test
  std::bitset<8> fPlaneMask;
     ///< Bitset for PlaneID used by algorithm, allowing for multiple planes and up to 8 total
          planes. Set to all 0s by default. Convention for bitset is that fPlaneMask[0] (i.e. bit 0)
          represents the collection plane, and then other planes work outwards from there.

  sParticleIDAlgScores(){
  fAlgName = "AlgNameNotSet";
  fVariableType = kNotSet;
  fTrackDir = kNoDirection;
  fAssumedPdg = 0;
  fNdf = -9999;
  fValue = -9999.;
  // fPlaneMask will use default constructor: sets all values to 0
  }
};
```

Particle ID Algorithm Struct

```
struct sParticleIDAlgScores { ///< determined particle ID
  std::string fAlgName;
     ///< Algorithm name (to be defined by experiment). Set to "AlgNameNotSet" by default.
  kVariableType fVariableType;
     ///< Variable type enum: defined in ParticleID_VariableTypeEnums.h. Set to kNotSet by default.
  kTrackDir fTrackDir;
     ///< Track direction enum: defined in ParticleID_VariableTypeEnums.h. Set to kNoDirection by
          default.
  int fNdf;
     ///< Number of degrees of freedom used by algorithm, if applicable. Set to -9999 by default.
  int fAssumedPdg;
     ///< PDG of particle hypothesis assumed by algorithm, if applicable. Set to 0 by default.
  float fValue; ///< Result of Particle ID algorithm/test
  std::bitset<8> fPlaneMask;
     ///< Bitset for PlaneID used by algorithm, allowing for multiple planes and up to 8 total
          planes. Set to all 0s by default. Convention for bitset is that fPlaneMask[0] (i.e. bit 0)
          represents the collection plane, and then other planes work outwards from there.

  sParticleIDAlgScores(){
  fAlgName = "AlgNameNotSet";
  fVariableType = kNotSet;
  fTrackDir = kNoDirection;
  fAssumedPdg = 0;
  fNdf = -9999;
  fValue = -9999.;
  // fPlaneMask will use default constructor: sets all values to 0
  }
};
```

- Key elements.
  - Algorithm name.
  - Value.
  - Plane mask.

Comparison of New vs. Old anab::ParticleID

- Advantages.
  - Extensible.
    - New particle id algorithms can be added without modifying the anab::ParticleID class.
  - Possibility of having particle id algorithms based on multiple planes.
- Disadvantages.
  - Assumes one TPC.
    - No cryostat id or tpc id.

The I/O Rule Saga

- The reason why the anab::ParticleID update was never merged into the develop branch is lack of working ioread rule.
  - Originally, it was not possible to create an ioread rule due to a root bug (lack of support for bitset template class).
  - The original root bug (if it existed) has long since been fixed.
  - A second issue was inability to read old geo::PlaneID.
    - Second issue is now solved (worked around).

How Root I/O Rules Work

- When reading a class using an ioread rule, root presents you with two versions of the object being read.
  - First version matches compiled in class.
    - Available as correct type pointer (T*) or reference (T&).
      - Class does not need to derive from TObject.
    - Layout matches global root dictionary.
    - Normal automatic schema evolution rules apply.
  - Second version is a reconstituted version of object read from disk, based on dictionary stored in disk file.
    - Available only as TObject*.
    - No programmatic access to class data members (can't downcast).
    - Class layout specified using object-specific dictionary (disk dictionary).
  - Task of ioread rule is to grab data from second object and update first object.

The geo::PlaneID I/O Rule Problem

- When reading an old version of anab::ParticleID from disk, second (TObject*) version contians an embedded geo::PlaneID object that needs to be unpacked in order to correctly set the plane mask in the new version of anab::ParticleID.
  - It has been observed that the geo::PlaneID object embedded in the reconstituted disk version of the object presented to the ioread rule is always default-constructed (invalid).
    - Possibly because of a problem with disk-resident class dictionary.
    - Problem of extracting old geo::PlaneID from disk TObject was never solved.
    - Nevertheless, the geo::PlaneID is actually in the data, because it can be read correctly using normal root I/O (without schema evolution).

Solving the I/O Rule Problem

- The solution of how to read the geo::PlaneID from old objects is to add a geo::PlaneID data member back in class anab::ParticleID (originally suggested by Philippe Canal).

```
class ParticleID{
private:

  std::vector<sParticleIDAlgScores> fParticleIDAlgScores;
            ///< Vector of structs to hold outputs from generic PID algorithms
  geo::PlaneID fPlaneID;   ///< Plane id.
```

  - Added data member must have type geo::PlaneID, and must have name fPlaneID to trigger default schema evolution.
  - Root's default schema evolution correctly fills fPlaneID data mamber in first object, even if class version and layout has changed.
  - Adds a place to store one cryostat id and tpc id in the bargain.
  - Other data members (than geo::PlaneID) are able to be read from disk TObject.
  - Updated ioread rule adds nine algorithm structs to anab::ParticleID, corresponding to nine atomic values in old anab:;ParticleID.

Larsoft anab::ParticleID Dependencies

- lardataobj.
  - ParticleID class.
  - ParticleID enum header (added).
  - classes_def.xml (ioread rule).
- lardata
  - DumpParticleIDs_module.cc
- larreco
  - KalmanFilterFitTrackMaker_tool.cc (pdg accessor).
- lareventdisplay
  - AnalysisBaseDrawer.cxx (commented out).
- larana
  - Chi2PIDAlg algorithm class.
  - Chi2ParticleID_module.cc

Larsoft Merge Branches

- Larsoft anab::ParticleID merge branches available in github.com/uboone.
  - lardataobj: greenlee_mcc9_pid
  - lardata: greenlee_mcc9_pid
  - larreco: greenlee_mcc9_pid
  - lareventdisplay: greenlee_mcc9_pid
  - larana: greenlee_mcc9_pid
- Based on revisions cherry-picked from MCC9 branches, originally based on merge branches supplied by authors Kirsty Duffy and Adam Lister.
  - Updated to be merge-compatible and buildable in recent integration releases (tested up to v09_34_00).

Experiment anab::ParticleID Dependencies

- There are known anab::ParticleID dependencies in many experiment-specific packages.
  - Many of these dependencies are related to experiment-specific versions of Analysis Tree ntuples.
- The original authors Kirsty Duffy and Adam Lister supplied merge branches with experiment-specific updates for the following packages.
  - argoneutcode: feature/kduffy_updatePIDdataprod
  - dunetpc: feature/kduffy_updatePIDdataprod
  - icaruscode: feature/kduffy_updatePIDdataprod
  - lariatsoft: feature/kduffy_updatePIDdataprod
  - sbndcode: feature/kduffy_updatePIDdataprod

Experiment anab::ParticleID Dependencies II

- Recommended way to merge experiment-specific branches to avoid conflicts.
  - git merge -X ignore-all-space <remote>/kduffy_updatePIDdataprod
- Additional caveats.
  - Lariatsoft appears to be unmaintained now.
  - Package sbncode has anab::ParticleID dependencies, but no existing merge branch.
  - Some experiment repositories have migrated to github. Merge branch may or may not have been migrated from redmine.
  - Experiment branches not fully tested.

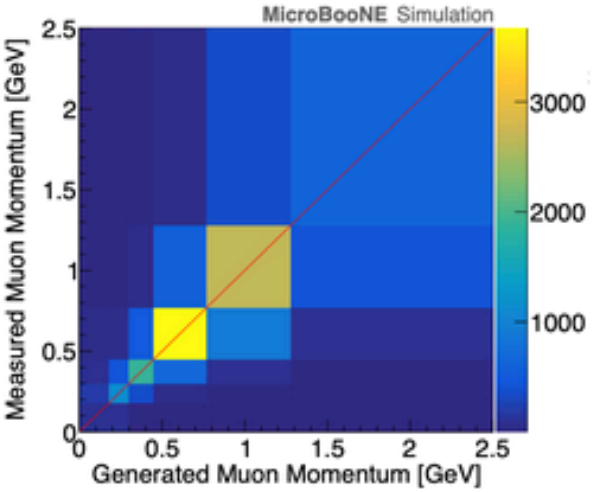
Additional MCC9 Updates - MCS Fitter

- Author Giuseppe Cerati.
- Larreco updates:
  - TrajectoryMCSFitter algorithm class.
  - mcsfitproducer.fcl
  - MCSFitProducer_module.cc not updated.
- Larreco merge branch: greenlee_mcc9_mcs

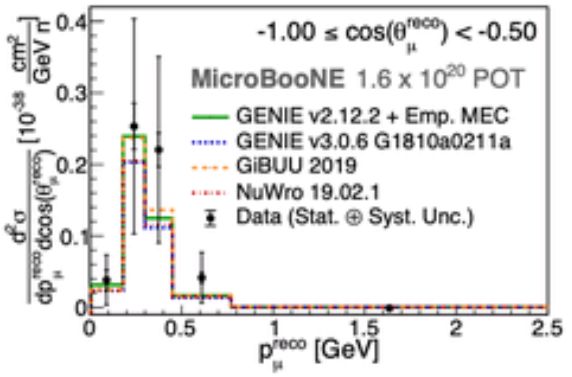
Giuseppe Cerati

MCS momentum track fit in LArSoft

- MCS momentum track fit split the track into segments and performs a likelihood scan over scatterings angles defined between segments thus fitting for the initial track momentum
  - only method available to estimate energy of exiting tracks
- Description in MicroBooNE paper JINST 12 (2017) 10, P10010
  - performance in data studied and presented in MICROBOONE-NOTE-1049-PUB
  - used in several physics analyses, e.g. CC inclusive paper
- Fitter code in LArSoft is in larreco/RecoAlg/TrajectoryMCSFitter
  - called from larreco/TrackFinder/MCSFitProducer_module
  - output stored in lardataobj/RecoBase/MCSFitResult.h



Phys. Rev. Lett. 123, 131801 (2019)



Updates to MCS algorithm

- Update to MCS momentum fit code from MicroBooNE MCC9 development
  - larreco branch greenlee_mcc9_mcs
- Several improvements to the algorithm, all configurable so previous behaviour is retained:
  - Perform a double raster scan, first a coarse one over full range and then a fine grained one around the minimum
  - Parametrization of Highland formula defined at fhicl file level
    - previously hardcoded, from uB paper
  - Add configurable parametrization for detector angular resolution, dependent on u_z
    - previously a constant value, defined in fhicl file
  - Optionally correct trajectory point positions for space charge effect based on available SpaceChargeService
  - Introduce a tolerance in segment length definition

Additional MCC9 Updates - Pandora Event Building

- Authors Andy Smith and Wouter van de PontSeele.
- Larpandora updates:
  - Updates limited to directory larpandora/LArPandoreEventBuiding.
  - CollectionMerging_module.cc removed.
  - CollectionSplitting_module.cc modified.
  - LArPandoraExternalEventBuilding_module.cc modified.
  - Associated tools and fcl files added and modified.
- Larpandora merge branch: greenlee_mcc9_event_building
- Event building updates were designed to be non-breaking for non-MicroBooNE experiments at the time they were originally developed (around Feb. 2019).

What MicroBooNE is Requesting

- We would like there to be a larsoft test build that incorporates the branches mentioned on the previous slides (repeated below) pertaining to ParticleID and other MCC9 updates.
- Branches summary.
  - lardataobj: greenlee_mcc9_pid
  - lardata: greenlee_mcc9_pid
  - larrecoj: greenlee_mcc9_pid
  - lareventdisplay: greenlee_mcc9_pid
  - larana: greenlee_mcc9_pid
  - larreco: greenlee_mcc9_mcs
  - larpandora: greenlee_mcc9_event_building
- All branches available in github.com/uboone.