Track design discussion

Erica Snider
Fermilab

March 31, 2014

Goals for this meeting

- To review the requirements for the Track class, including the newly proposed attributes
- To review the interface and abstractions used
- To discuss the role of BezierTrack
- To propose changes needed to address any issues
- Discuss other associated issues

What is a track?

- A complex collection of data, parameters and attributes. Broadly speaking:
  - A trajectory through space representing the path a particle took
  - A set of hits that represent the measurement positions of the particle.
  - The ionization energy deposited by the particle at various points along the traj.
    - The same as the position measurements in LArSoft, but logically can be treated as separate from the position estimates.
  - A set of parameters that define the trajectory at one or more points
    - Includes our knowledge of how well we know the parameters and the fit quality, NDoF
    - Should include information on which hits are included in the fit, and how, if important
  - Momentum / energy
    - Might require a PID. Probably also need to store the algorithm and uncertainties
  - A collection of other useful attributes
    - Geometrical configuration (ie, exits the side, through-going, etc)
    - Number of hits
    - PID / PID probabilities
    - Algorithm tag
    - Parent track(s) pointer

What is a track?

- They come from many different sources and times in the reconstruction chain:
  - A trajectory through space representing the path a particle took
    - Output of a fitting algorithm or some other trajectory estimation algorithm
  - A set of hits that represent the measurement positions of the particle.
    - Output of a pattern-recognition algorithm
  - The ionization energy deposited by the particle at various points along the traj.
    - The product of an energy estimation algorithm. May or may not be associated with the position estimation procedure.
  - A set of parameters that define the trajectory at one or more points
    - Output of a fitting algorithm
  - Momentum / energy
    - Output of a calorimetry algorithm or range estimating technique.
  - A collection of other useful attributes
    - Geometrical configuration (ie, exits the side, through-going, etc)
    - Number of hits
    - PID / PID probabilities
    - Algorithm tag
    - Parent track pointer(s)

What is a track?

- One could imagine multiple versions of some attributes associated with the track:
  - A trajectory through space representing the path a particle took
    - If multiple fits
  - A set of hits that represent the measurement positions of the particle.
    - Fixed by definition
  - The ionization energy deposited by the particle at various points along the traj.
    - The result of different algorithms or calibration passes
  - A set of parameters that define the trajectory at one or more points
    - If multiple fits
  - Momentum / energy
    - Multiple versions out of the box
  - A collection of other useful attributes
    - Geometric configuration (possible if multiple fits allowed)
    - PID probabilities

Where does BezierTrack come in?

- Adds a continuous representation for the track trajectory
  - Uses sub-set of trajectory hits in parameterization of this trajectory
- A trajectory, continuous or otherwise, is an attribute of a track
  - There is not an “is-a” relationship between a trajectory and a track
- Suggest that we make a trajectory class that we add as a data member (or via association) to a track
  - Eliminate the BezierTrack class altogether

What next?

- First, what do we *need* to have associated with a Track object?
  - Are all these parameters really members of what we call a track?
  - Everything that is absolutely needed should be in the interface
    - Can use external associations for the rest (?)
- Second, how should we represent the data?
  - Some things may be better represented as associations with the “track”
    - Particularly those that are created at different times from the original track, or have multiple versions
    - “Association” here means “by reference”, so not strictly talking about art::Assn
  - Can hide complexity of associations behind interface of a unifying class
    - For example, could use the existing track interface + new accessors
    - The“facade”/helper class should provide access to the

A couple of broad design principles

- Data products should generally be simple data structures
  - Complex behavior should be delegated to associated helper classes that use / contain the data structures
- Inheritance among data products generally does not work well
  - Two problems
    - The persistency and event model schemes make it difficult to access by reference to the base class
      - Collections carry the type of the sub-class
      - Smart pointers carry the type of the sub-class
    - Inheritance deals with object behavior
      - Data objects, with rare exception, should have very little behavior
      - Other types of relationships can be handled equally well using something other than inheritance
  - Options are to make lots of different track classes, or make one that allows considerable flexibility in the data it provides
    - Associations can do this easily

A comment on using tracks

- Might be useful to adopt the use of reference lists to access tracks
  - Provides a simple mechanism for selecting and sorting tracks that are otherwise stored in immutable collections.
  - Provides a simple means of gathering tracks from multiple collections, as would occur if there were more than one tracking algorithm or pass.
  - Allows analyzers to define small collections of tracks associated with some process.
    - A vertex needs this type of list if that vertex is defined from tracks
- Can do this with art Assn, perhaps wrapped in something that makes it clear what it is.

Discussion

- Requirements
- Interface
- BezierTrack
- Changes
- Other issues