LArSoft Architecture Meeting

Erica Snider
Gianluca Petrillo
Fermilab

December 8, 2014

Agenda

- Proposed changes to low-level architecture
- Discussion that leads to approving, or actions / decisions needed first

Scope of discussion

- What we need to consider
  - Changes to address issues identified via internal review of data products, input from experiments (mostly uBooNE), discussion from previous meetings and in hallways
  - Restrict ourselves to low-level data products
    - Will address high-level data products later
- What we will not consider
  - Questions regarding which algorithms will actually be used
    - Just agree on the abstractions needed to support existing or anticipated algorithms
  - The effort required, resources available, schedules for the changes
    - This should be a separate discussion with experiment management
    - Now just need to decide on the eventual objectives

The “low-level” data products considered

- raw::RawDigit
- recob::Wire
- recob::Hit
- recob::Cluster
- recob::EndPoint2
- recob::Track

raw::RawDigit

- Proposed change
  - Remove pedestal value
    - If subtracted prior to building the object, then not needed
      - It's in a database or somewhere else anyway
    - If calculated on the fly, then it's not needed in the RawDigit
  - Introduce values + methods to flag if a channel is saturated
    - Needed in the case that RawDigit contains pedestal-subtracted values

Current interface:

```
public:

  std::vector<short> fADC;

  // Set Methods
  void             SetPedestal(double ped);

  // Get Methods
  unsigned int    NADC()        const;
  short           ADC(int i)    const;
  uint32_t        Channel()     const;
  unsigned short  Samples()     const;
  double          GetPedestal() const;
  double          GetSigma()    const;
  raw::Compress_t Compression() const;
```

recob::Wire

- Recently modified to accommodate regions of interest
  - This was an important, useful change
- Further change proposed
  - Store the channel number internally
  - Eliminate internally stored art::Ptr to RawDigit, replace with external association
  - Eliminate SignalType() method
    - Implicit in the plane, so needn't be in each Wire object

Current recob::Wire

```
public:

  // ROI constructor
  Wire(const RegionsOfInterest_t& sigROIlist,
       art::Ptr<raw::RawDigit> &rawdigit);
  Wire(RegionsOfInterest_t&& sigROIlist,
       art::Ptr<raw::RawDigit> &rawdigit);

  // Get Methods
  // zero-padded full length vector filled with ROIs
  std::vector<float>  Signal() const;

  const RegionsOfInterest_t& SignalROI()  const;
  size_t                     NSignal()    const;
  art::Ptr<raw::RawDigit>    RawDigit()   const;
  geo::View_t                View()       const;
  geo::SigType_t             SignalType() const;
  uint32_t                   Channel()    const;
```

recob::Hit

- Lots of ideas from people about what to do with hits
  - Differences between requirements of LBNE and uBooNE
    - uBooNE assumes recob::Wire is present, so suggests dropping duplicated data
    - LBNE does not store recob::Wire objects
  - This will limit the scope of proposed changes

Current recob::Hit

```
public:

    // Get Methods
    double                  StartTime()                 const;
    double                  EndTime()                   const;
    double                  PeakTime()                  const;
    double                  SigmaStartTime()            const;
    double                  SigmaEndTime()              const;
    double                  SigmaPeakTime()             const;
    int                     Multiplicity()              const;
    uint32_t                Channel()                   const;
    double                  Charge(bool max=false)      const;
    double                  SigmaCharge(bool max=false) const;
    double                  GoodnessOfFit()             const;

    geo::SigType_t          SignalType()                const;
    geo::View_t             View()                      const;
    art::Ptr<recob::Wire>   Wire()                      const;
    art::Ptr<raw::RawDigit> RawDigit()                  const;
    geo::WireID             WireID()                    const;
```

recob::Hit

- The issues
  - HitSignal vector duplicated by recob::Wire
  - Charge() method
    - Takes an argument bool to toggle between “max” and summed ADC charge
    - The “max” is really the “peak” of the fit, not the maximum ADC value observed
    - PeakTime is the time (in TDC ticks) associated with that estimated peak
  - Sigma() method: assumes Gaussian shape
  - Multiplicity() is the number of hits found between StartTick() and EndTick()
    - No index available to say where a hit is in the train of hits found
  - Internal art::Ptr to associated recob::Wire and raw::RawDigit objects
  - Policy issues
    - Clarify that “time” means “TDC tick”
    - StartTick() and EndTick() represent the interval over which hit-finding was performed, and are not start and end points for the individual hit.
      - Can therefore remove SigmaStartTime() and SigmaEndTime()

recob::Hit

- Proposed changes
  - Replace Charge() method with
    - PeakAmplitude(), which returns fitted peak ADC value
    - SummedADC(), which returns sum of ADC values apportioned appropirately between shared hits
    - IntegratedADC(), the integral of the fit, so the best estimate of collected charge
  - Replace Sigma() method with FWHM()
    - More general, so covers the case of non-Gaussian hit shapes
    - Allows simple calculation of Gaussian sigma when needed.
  - Replace StartTime() / EndTime() with StartTick() / EndTick() to clarify meaning
    - Remove SigmaStartTime() and SigmaEndTime()
  - Add `LocalIndex()`, which returns position of hit among those found within the StartTick() to EndTick() region, starting from StartTick() side.
  - Drop internal art::Ptr objects. Use external associations as needed.

recob::Hit

- Proposed changes (cont'd)
  - Leave HitSignal() as is
    - This is a large overhead for uBooNE, so may need a better solution

recob::Cluster

- Add methods used in shower versus track discrimination
  - NHits()
  - OpeningAngle(), a shape variable
  - TotalSummedADC()
  - AverageSummedADC()
  - RMSSummedADC()
  - `TotalIntegrateADC()`
  - NWire / NHit, a shower / track disciminant
  - Width(), a shape variable
- Remove dQ/dW(), which is unused

Current recob::Cluster

```
public:
  /// Accessors
  double                 Charge()        const;
  geo::View_t            View()          const;
  double                 dTdW()          const;
  double                 dQdW()          const;
  double                 SigmadTdW()     const;
  double                 SigmadQdW()     const;
  std::vector<double>    StartPos()      const;
  std::vector<double>    EndPos()        const;
  std::vector<double>    SigmaStartPos() const;
  std::vector<double>    SigmaEndPos()   const;
  int                    ID()            const;
  const geo::PlaneID&    Plane()         const; ///< returns the geometry plane of the cluster
  //@}

  /// Returns whether geometry plane is valid
  bool                   hasPlane()      const;

  /// Moves the cluster to the specified plane
  Cluster& MoveToPlane(const geo::PlaneID& new_plane);

  /// Makes the plane of this cluster invalid
  Cluster& InvalidatePlane();

  Cluster              operator +  (const Cluster&);
```

recob::EndPoint2D

- The issues
  - We need two distinct roles
    - A point in a plane indicating, e.g., the start / end of a cluster (a geometric object)
    - A 2D vertex (a reconstructed object)
- Proposed changes
  - Use a 2D point in places where the geometry object is needed
  - Introduce a 2D reconstructed vertex class for cases when vertex is needed
    - Move Strength() method here
    - Add Multiplicity() to indicate how many clusters / tracks are associated

Current recob::EndPoint2D

```
public:
  EndPoint2D(double driftTime,
             geo::WireID wireID,
             double strength,
             int id,
             geo::View_t view,
             double totalQ);

  double          Charge()    const;
  geo::View_t     View()      const;
  double          DriftTime() const;
  geo::WireID     WireID()    const;
  int             ID()        const;
  double          Strength()  const;
```

recob::Track

- The main issues
  - A track is a collection of attributes of the clusters / collection of hits that form the track + a trajectory + results of a fit used to obtain the trajectory
    - Anything else is typically computed separately, so should be associated
      - Momentum estimate, energy estimate, PID, etc
  - Bezier tracks:
    - Parameterization for a continuous trajectory stored as recob::Track trajectory pts
      - Such tracks cannot be interpreted like a “normal” recob::Track
    - The actual BezierTrack class inherits from Track, is created as a transient object from a recob::Track
  - Meaning of trajectory points is not clear
    - Meaning, method of defining trajectory points are not well-defined
  - Contains dQ/dx
    - Not used anywhere, and not needed as part of pattern recognition

Current recob::Track

```
public:

  void            Extent(std::vector<double> &xyzStart,
                         std::vector<double> &xyzEnd)        const;
  void            Direction(double *dcosStart,
                            double *dcosEnd)                 const;
  double          ProjectedLength(geo::View_t view)          const;
  double          PitchInView(geo::View_t view,
                              size_t trajectory_point=0)     const;
  int             ID()                                       const;


  // A trajectory point is the combination of a position vector
  // and its corresponding direction vector
  size_t          NumberTrajectoryPoints()                    const;
  size_t          NumberCovariance()                          const;
  size_t          NumberFitMomentum()                         const;
  size_t          NumberdQdx(geo::View_t view=geo::kUnknown)  const;
  double          Length(size_t p=0)                          const;
  void            TrajectoryAtPoint(unsigned int  p,
                                    TVector3     &pos,
                                    TVector3     &dir)        const;
  const double&   DQdxAtPoint(unsigned int p,
                              geo::View_t view=geo::kUnknown) const;
  const TVector3& DirectionAtPoint (unsigned int p)           const;
  const TVector3& LocationAtPoint  (unsigned int p)           const;
  const double&   MomentumAtPoint  (unsigned int p)           const;
  const TMatrixD& CovarianceAtPoint(unsigned int p)           const;

  const TVector3& Vertex()                                    const;
  const TVector3& End()                                       const;
  const TVector3& VertexDirection()                           const;
  const TVector3& EndDirection()                              const;
  const TMatrixD& VertexCovariance()                          const;
  const TMatrixD& EndCovariance()                             const;
  const double&   VertexMomentum()                            const;
  const double&   EndMomentum()                               const;

  double          Theta()                                     const;
  double          Phi()                                       const;

  // Calculate rotation matrices between global (x,y,z) and local (u,v,w)
  // coordinate systems based on track direction (fDir).
  // The local w-axis points along the track direction.
  void GlobalToLocalRotationAtPoint(unsigned int p, TMatrixD& rot) const;
  void LocalToGlobalRotationAtPoint(unsigned int p, TMatrixD& rot) const;
```

recob::Track

- The proposed solution (one of many possible)
  - Introduce a trajectory class to represent continuous trajectory
    - Abstract interface + concrete data product classes for specific parameterizations
    - Classes that now use BezierTrack should use recob::Track + trajectory class
  - Introduce a momentum object
    - Contains vector of momentum vectors, covariances + other parameters needed to characterize quality of the momentum estimate
    - Typically only care about the momentum at the vertex
      - Do not require that N momentum estimates = N trajectory points
  - Remove fit momentum from the track
    - Currently not used anywhere
    - But, If needed for physics, create a momentum object
  - Remove dQ/dx
    - Not used, and is a calorimetry object anyway

recob::Track

- The proposed solution (cont'd)
  - Trajectory points
    - Much thought and discussion about whether to include these in the track or put them into an associated object
      - Include:
        - *Most people consider them to be an intrinsic property of the track*
        - *Easy to use if just in there*
      - Associate:
        - *Could be multiple ways to calculate them, particularly if standardizing on a definition*
    - Eventually decided to leave them in the track for now
      - Define a policy / algorithm for calculating the trajectory points
      - Wire plane intersections,
        - *If distance between points is > a configurable maximum, add mid-point*

        This can be discussed separately

Discussion