

Reconstruction Code

LArSoft Meeting
Mitch Soderberg
October 15, 2009

Intro

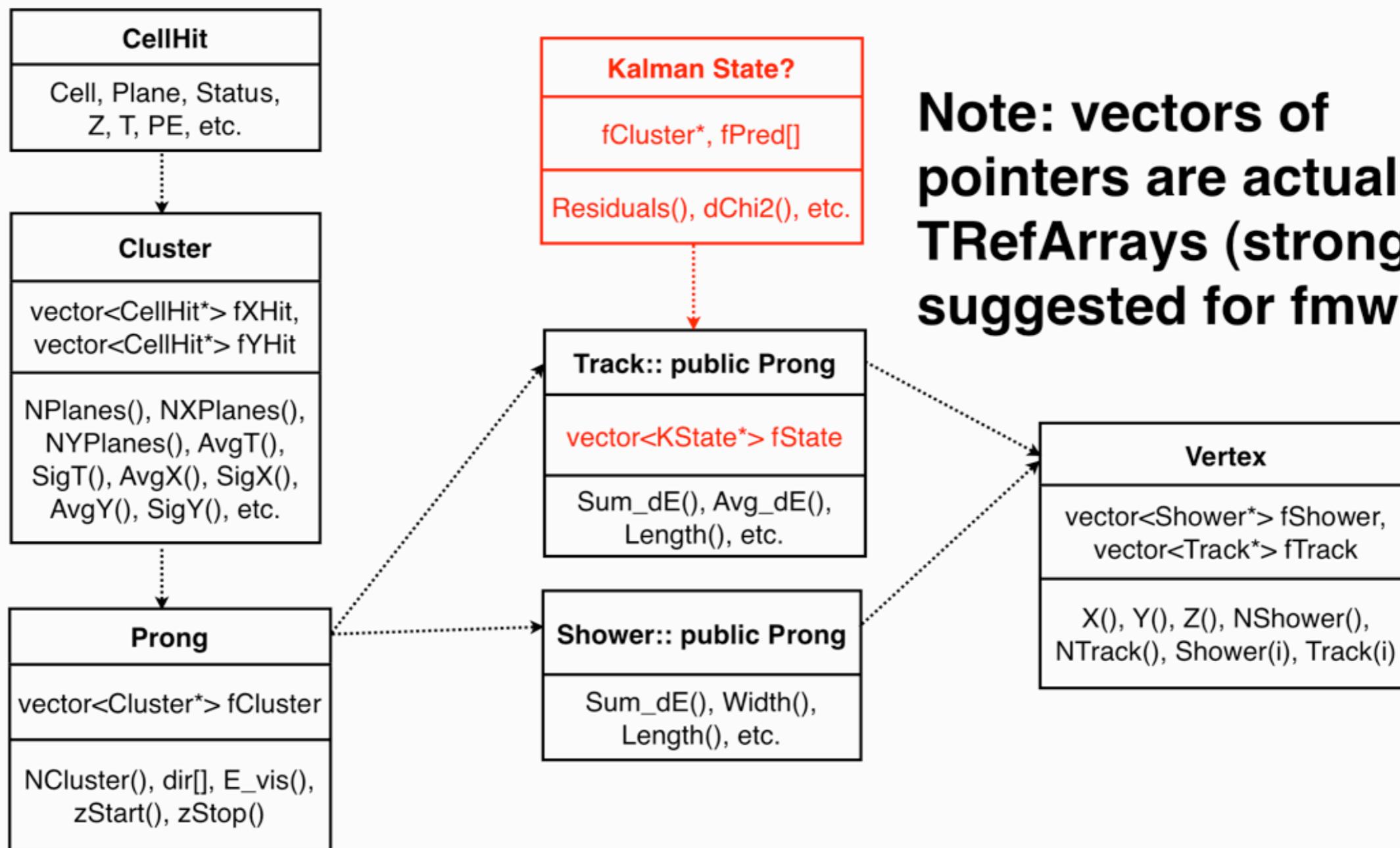
- We've been discussing what to do about Reconstruction code for a few months...need to make some decisions and start coding.
- Before we jump into coding algorithms for new objects (tracks, showers, etc...), we need to discuss the definition and function of the objects that make up the Reconstruction classes.
 - ▶ We don't want to have to redo all of this 6 months from now!
 - ▶ Generic Issues:
 - ◆ What are the basic Reconstruction classes we think we need?
 - ◆ How should these classes be implemented within the FMVVK/LArSoft framework?
 - ◆ What data members/methods are desired for the various classes?
 - ◆ How is the class information accessed by users and other pieces of code?
- Tangents not specific to Reconstruction:
 - ▶ How will people actually access data and run jobs?
 - ▶ What other items are in need of discussion?

Reconstruction

- Reconstruction refers to all the tools we use to identify and characterize the important features of our raw data.
- Our algorithms for identifying these features will evolve over time, but to make some coding progress we need to define an initial set of objects that will contain the relevant information necessary for analysis.
- Creating these classes is not necessarily hard, but we may stick with them for a long time, so let's give it some thought.
- “Hit” class is one example of an existing reconstruction class, but it is still in need of some fine tuning.

Example: NOvA Reconstruction

General Overview



Note: vectors of pointers are actually TRefArrays (strongly suggested for fmwk).

RecoBase Package: Hit Class

- Hit class already exists in LArSoft RecoBase package
 - ▶ Fairly minimal data members right now (fTotalSignal, fStartTime, fCenterTime, fEndTime, fTransversePos, fHitSignal, fWireRef)
 - ▶ No way to save parameter values (if doing shape fitting type hit-finding).
 - ▶ A 2D object as written (i.e. - wire # and hit time are only spatial information).
 - ▶ Interface to EventDisplay is pretty simple right now...can overlay Hits, but can't really do much else with them (click on them to get info., display their information, etc...). This is more of an EventDisplay issue.

```
class Hit : public TObject {
public:
    Hit(); // Default constructor
    Hit(double totsigt,
         double startT,
         double centerT,
         double endT,
         double transversePos,
         std::vector<double> signal,
         const recobase::Wire *wire);

    ~Hit();

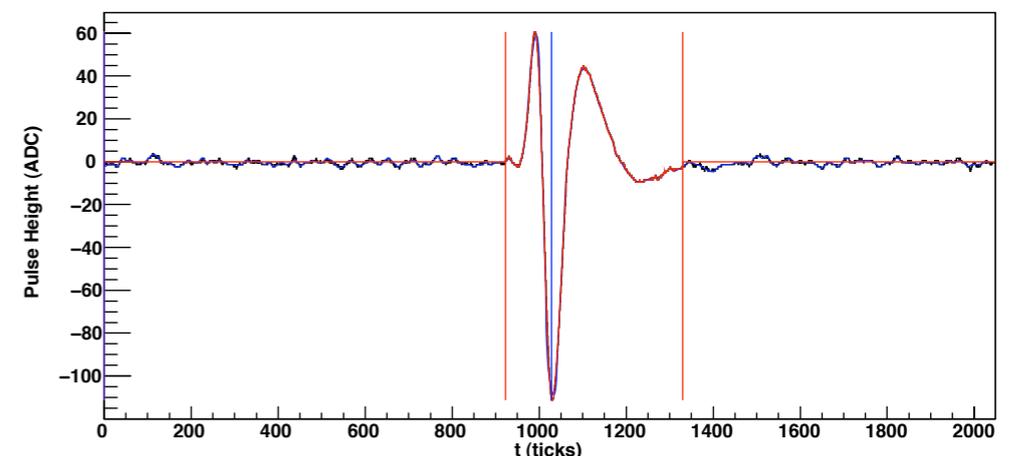
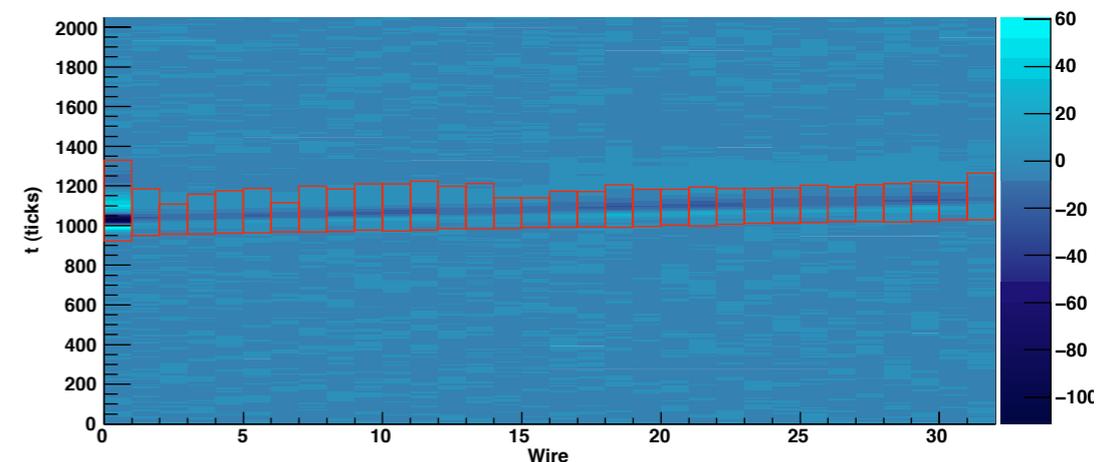
    // Set Methods
    void SetTotalSignal(double totsigt) { fTotalSignal = totsigt;}
    void SetStartTime(double startT) { fStartTime = startT;}
    void SetEndTime(double endT) { fEndTime = endT;}
    void SetCenterTime(double centerT) { fCenterTime = centerT;}
    void SetTransversePos(double tPos) { fTransversePos = tPos;}

    // Get Methods
    double GetTotalSignal() const { return fTotalSignal;}
    double GetStartTime() const { return fStartTime;}
    double GetEndTime() const { return fEndTime;}
    double GetCenterTime() const { return fCenterTime;}
    double GetTransversePos() const { return fTransversePos;}
    recobase::Wire *GetWire() const { return dynamic_cast<recobase::Wire *>(fWireRef.GetObject());}
    std::vector<double> fHitSignal;

private:
    double fTotalSignal;
    double fStartTime;
    double fCenterTime;
    double fEndTime;
    double fTransversePos;

    TRef fWireRef;

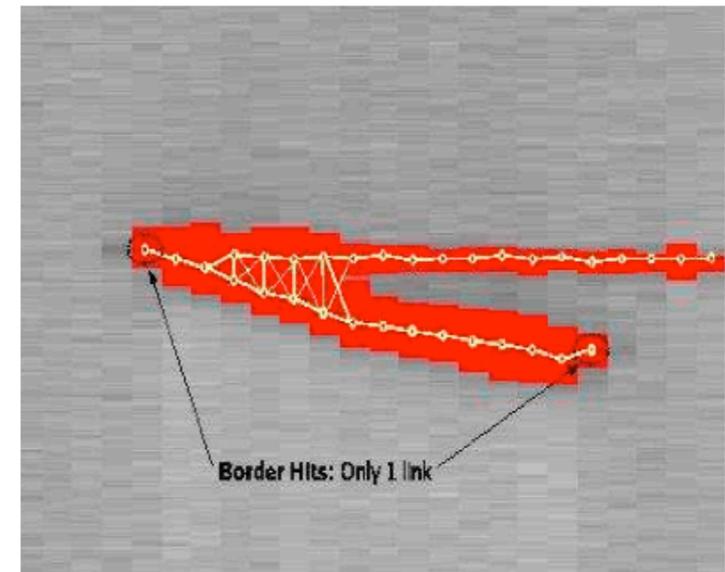
    ClassDef(Hit, 3) // Hit recobase object
};
```



Revising Hit Class

- Some time ago I made a first pass at a revised “Hit” Class

- ▶ Constructor now only takes a string (for the HitFinding algorithm/collection name) and a TRef to the Wire object the Hit was created from.
- ▶ Members: added more members for the details of the hit object (max./min.ADC value, crossing time, max./min.ADC time sample, MIP equivalent, etc...)
- ▶ Added a vector of TRefs where “linked” Hits can be stored.
- ▶ Added a TRef for a Cluster object that the Hit belongs to.
- ▶ Added a map of <string,double> that can be used to store additional parameters and their values, as needed by different algorithms
- ▶ Added functions to list/return parameter names/values.
- ▶ Added methods to add/subtract Links/Clusters
 - ◆ Link = (a la ICARUS) hits on same/neighboring wire that are close spatially
- ▶ Added Print method.



- Brian pointed out that we might reconsider storing parameter/collection names with every Hit object...might consider using Header class or some other database interface.

- These issues will translate to all other RecoBase classes.

Algorithms for Creating Collections

- Along with creating RecoBase classes, need to think about how they get filled.
- Initial idea: For every step in the reconstruction chain, there is an object in the RecoBase package, and a corresponding package with algorithm code.
- Example: for the Hit class I created a package called HitFinder:
 - ▶ Contains base class (HitFinder) which handles mundane stuff (fetching desired objects, writing out new objects, etc...)
 - ▶ Users create inheriting classes (ExampleHitFinder) which only have to implement a specific algorithm (called FindHit) for creating the Hit objects.
 - ▶ Parameters specific to the individual algorithms can be accessed via XML if user adds necessary code to the Update function.
 - ▶ Better idea? - define a single RecoBase algorithm that generically handles mundane stuff...make all specific algorithms inherit from this class. Would save the need for individual base classes which are more or less identical, except for their specific input/output types.

Algorithm	RecoBase Object	larsoft Package
Deconvolution/Calibration	Wire	CalData
Hit Finding	Hit	HitFinder
Segment Finding	Segment	SegmentFinder
Track Finding	Track	TrackFinder
Vertex Finding	Vertex	VertexFinder
Shower Finding	Shower	ShowerFinder
Event Summary	Event	EventBuilder
Others	Others	Others

Don't exist yet!

Job Flow

- This structure cleanly allows for different implementations of a reco. algorithm (e.g. - FFTHitFinding vs. DifferenceHitFinding) to be run successively on a given event.
- Consider one imaginary job sequence:

Algorithm	Input/Output Folders	Purpose
Calibrate	raw digits/wires1	Create Calibrated Wires
Deconvolute1	wires1/wires2	Perform deconvolution
Deconvolute2	wires1/wires3	Alternative deconvolution
WindowHitFinder	wires2/hit1	Find Region of Interest
FFTHitFinder	hit1/hit2	Fancier hit finding
SegmentFinder	hit2/segment	Group hits
MitchTracker	segment/track1	Group segments
BrianTracker	segment/track2	Group segments
PrimaryVertex	track2/vertex	Group tracks
ShowerFinder	?/shower	Cluster hits/tracks
EventBuilder	?/event	Summarize

- Note: FMWK can already do this type of sequencing without all this “base class” stuff....but implementation of classes is not as compact/clean.
- Question? Do we intend to stick with the “Folder” style of output (i.e. - each new collection gets put in it’s own ROOT folder). How is this handled when more info. gets added?

New Base Classes?

- Can try and copy what other experiments already do....
- Cluster/Prong/Vertex classes not yet in LArSoft
 - ▶ A Cluster is a grouping of “related” Hit objects.
 - ▶ Definition of the sufficient relation must be defined...can utilize Links from the Hit objects.
 - ▶ A Prong is a grouping of “related” Cluster objects.
 - ▶ Tracks and Showers are special cases of Prongs....perhaps don't need their own classes?
 - ▶ Vertex is a grouping of “related” Prong objects.
 - ▶ Could follow the same idea implemented in HitFinder...define generic base class for Cluster/Prong/Vertex finding, then put specific algorithms in inheriting classes?
- There are probably numerous other Reco. classes I have not mentioned (PMT related objects, etc...).
- I had started to write simple classes for storing P.O.T. information, and other such run time parameters....not strictly a Reconstruction issue, but does have some impact and should be part of the discussion.

Tangents

- How will people access data and run jobs?

- ▶ We are already having trouble with people running out of disk space, or wanting to know how to access the ArgoNeuT data.
- ▶ We have BlueArc space in the works, as well as Enstore space, but we don't really have a scheme for using it effectively .
- ▶ Do we have any plan for expanding computing resources?

- What other items are in need of discussion?

- ▶ Access to simple databases from within LArSoft would be very useful...I see a little evidence that there is database related code in FMWK...has anyone ever used it?
- ▶ The EventDisplay program has served us well to this point, but it needs more work before it's as useful to non-experts as it could be. (i.e. - clickable objects, more control over the display settings, better use of space, standardized aspect ratios and the like, etc...).