Wire Cell Software Overview and Status

Brett Viren

Physics Department

BROOKHAVEN
NATIONAL LABORATORY

BNLIF Wire Cell Team
2015 Aug 11

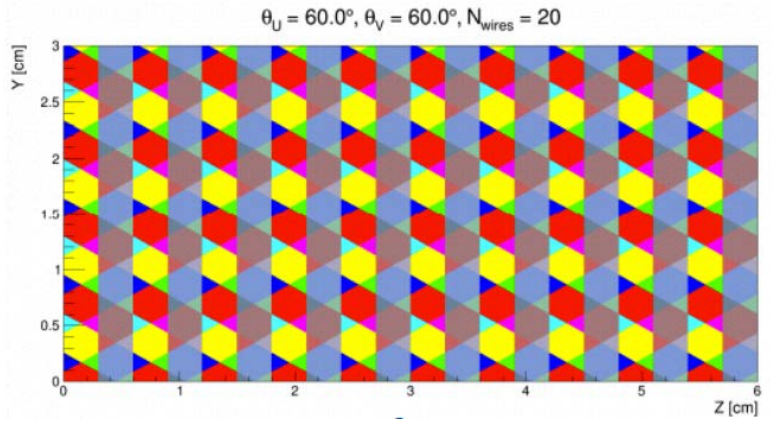
Outline

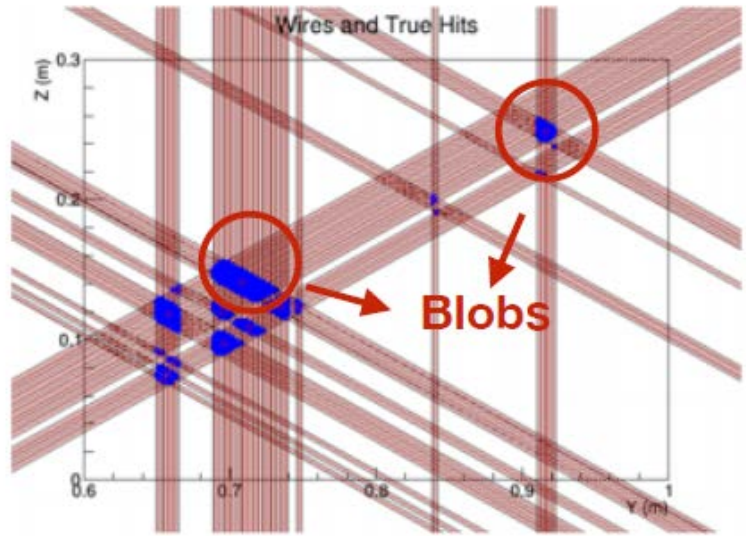
Method

Wire Cell Software

Connections with LArSoft

Wire Cell Gestalt



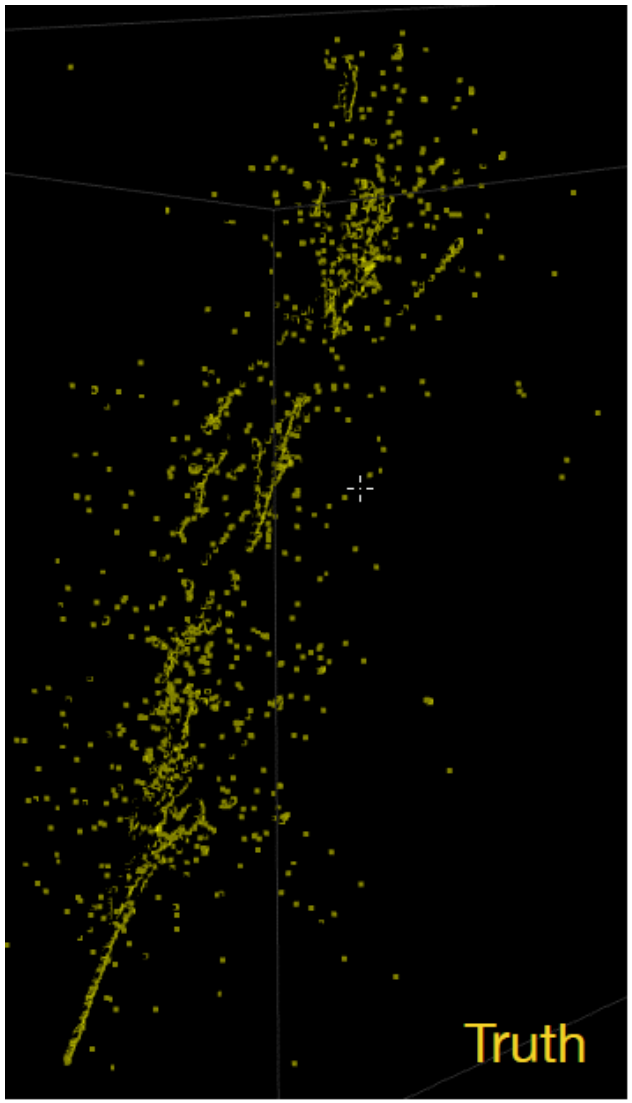


1. Tile the wire plane with "cells", each associated with one wire from each plane.
2. Focus on a time-slice across the readout channels (nominally 4 ticks).
3. Determine which cells may contain charge consistent with the sliced readout for their wires.
4. Merge potential hit cells into "blobs" to reduce multiplicity.
5. Attempt to solve a wire-cell association matrix via χ^2 minimization.
6. Associate solution back in time to drift origin to form 3D point.
7. Clustering, PID, Physics!

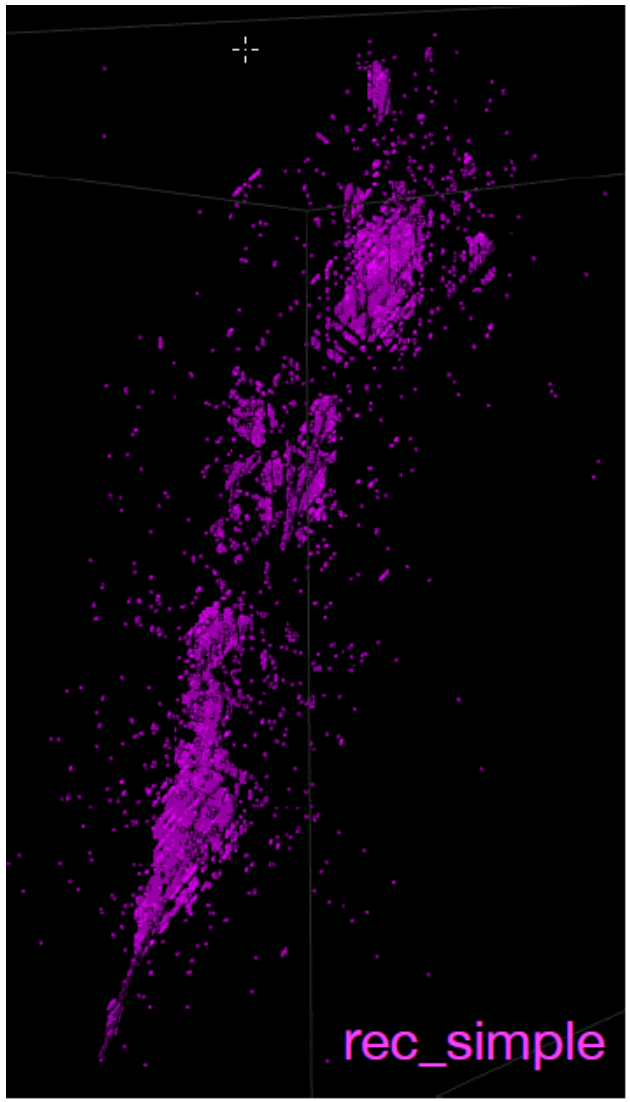
For more information, see:

- Chao Zhang's presentation to 35t/FD Sim, Reco, and Analysis.
- A detailed paper is in preparation.

Wire Cell Example Solution, 1.5 GeV e^-



MC depositions.



Only geometry

Geometry and charge.

Wire Cell Software Ecosystem

Consists roughly of these parts:

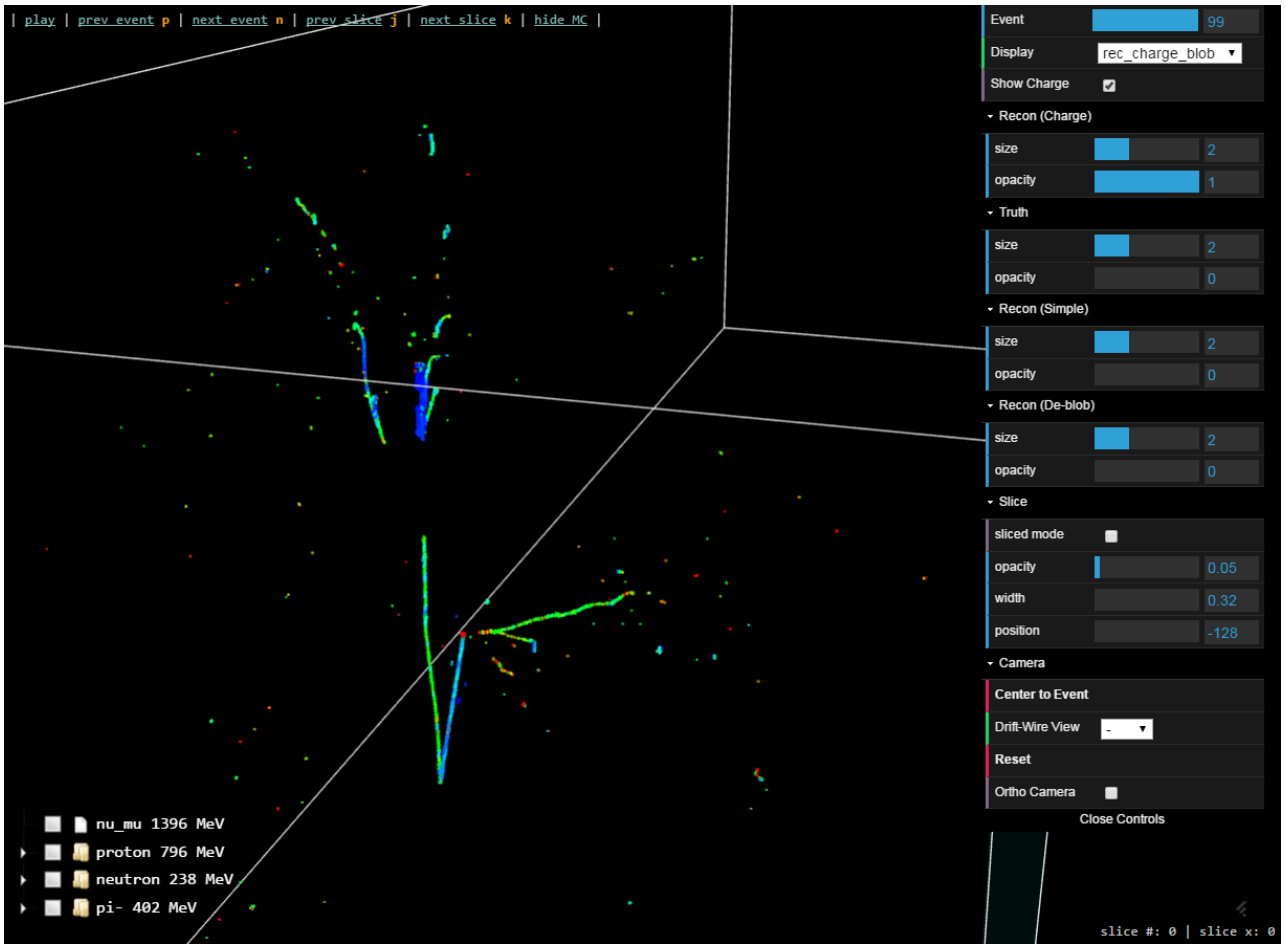
bee 3D interactive web-based event display.

core C++ libraries providing:

- data and geometry representations.
- the reconstruction procedures themselves.
- charge drifting and signal processing.
- various data/geometry file I/O.
- job configuration and a processing model.

misc file-based data exchange with LArSoft, stand-alone signal processing studies, experimental clustering, bee file server, some others.

Bee



http://www.phy.bnl.gov/wire-cell/examples/mvd/numu-nc-v2/#/99

- 3D, interactive, web-browser interface using WebGL acceleration.
- Display MC truth, WC and other reconstruction results, simple JSON file format.
- Server side file organization and JS delivery based on Django.

Wire Cell Source Code Repositories

The Bee display:

- wire-cell-viz-webgl repository.
- Primary developer: Chao.

Wire Cell "core" repositories have **two active branches**:

master **working prototype** code producing the results we've been showing, (primary: Xin).

ifaceio fork of `master`, for **structured, production code** for long term development, tuning and toward supporting parallel architectures, (primary: bv).

Focus on Wire Cell "core" parts next →

Installation of Wire Cell "core"

Primary dependencies (subject to change):

build:	C++11 compiler (GCC 4.9.2 used)
core:	Boost (1.55)
apps/tests:	ROOT 6 (6.05/01)

Some build details:

- Source packages aggregated via `git submodules`
- Native build system: waf (self-contained copy provided).
- Unit and integration tests run regularly as part of the build.
- Installs shared libraries + headers + few main applications.

Details at `http://bnlif.github.io/wire-cell-docs/install/`.

Wire Cell "core" Packages Overview

Wire Cell core packages, most named like `wire-cell-*`:

- **wire-cell** `git submodule` aggregation and top-level build-package, (`master`, `ifaceio`).
- **-data** common, concrete data classes, (`master`).
- **-2dtoy** working prototype implementation, (`master`).
- **-util** general utility code, 3D vector, system of units, configuration files, various C++ patterns, (`ifaceio`).
- **-iface** interface classes for major components and data classes, (`ifaceio`).
- **-nav** default implementation of wire cell components with minimal outside dependencies, (`master`, `ifaceio`).
- **-rio** internal ROOT I/O persistency, (`ifaceio`).
- **-sst** "simple simulation tree" file-interface to use LArSoft data and geometry as one possible input to Wire Cell, (`master`, `ifaceio`).
- **waf-tools** `waf` support files for the native build system, (`master`, `ifaceio`).

All found at `https://github.com/BNLIF/`.

Wire Cell Class Interfaces

- All important Wire Cell classes (in `ifaceio`) inherit from abstract base classes following **Interface** patterns.
- Interface methods take POD or Interfaces.
- Code is in the `wire-cell-iface` package.
- `std::shared_ptr<>` memory ownership rules,

Two main categories of interfaces:

data (eg, `ICell`, `ISlice`)

- Data is const once created.
- Abstract iterator interface for collections.

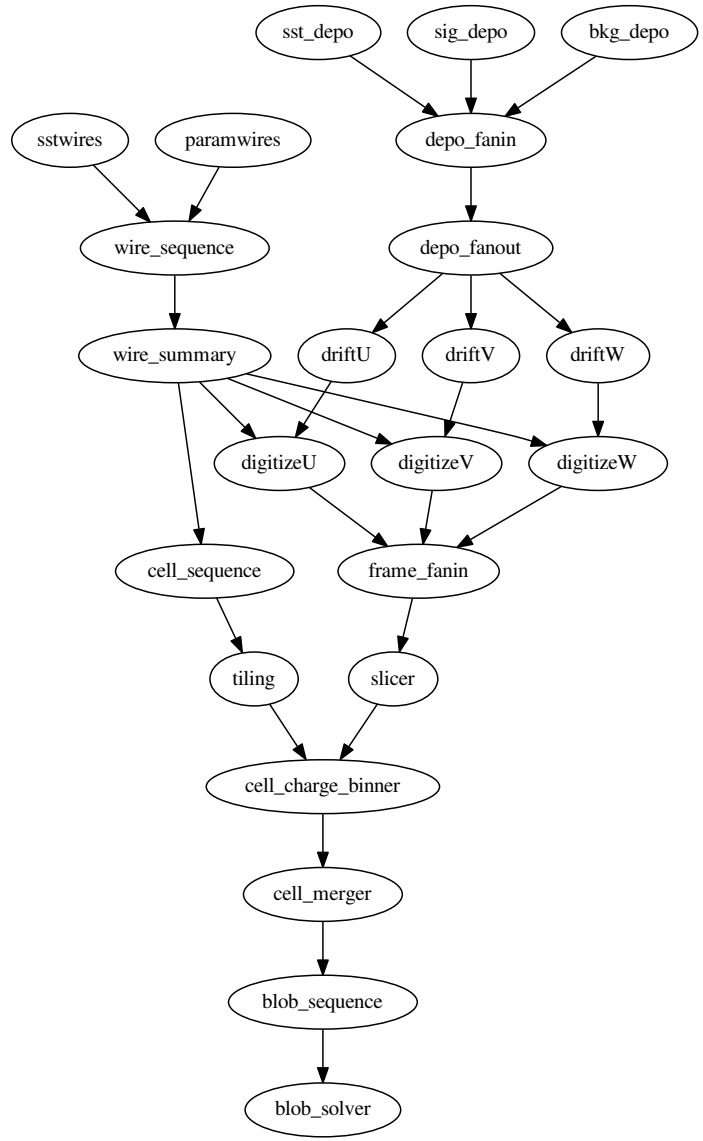
component (eg, `IConfigurable`, `IWireSummary`, `ITiling`)

- Interface defines a facet of functionality.
- Name-based location/construction to support job configuration and loosely coupled build dependencies.

Data Flow Programming Paradigm

Wire Cell supports the "data flow programming" paradigm.

- Components are written to provide well formed "sockets" (methods) connected to accepting "signals".
- Connections prototypes are standardized based on purpose.
- Execution model is synchronous "pull".
- Potential for fine-grained parallel execution model with no component code changes.
- Now, connections formed in C++ but this will be exported to the end-user configuration layer.

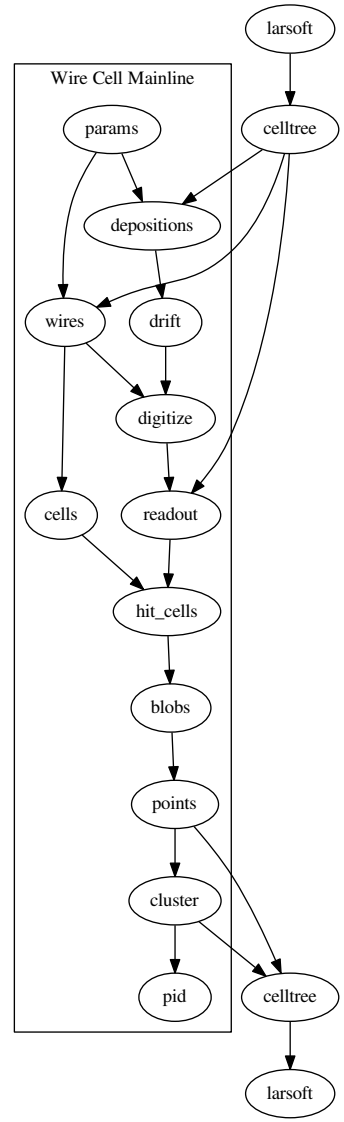


Current Wire Cell / LArSoft Integration

- Wire Cell (core) is wholly independent from LArSoft.
- Stand-alone deposition, drifting, digitizing, etc provided.
- Deposition/digit/geometry input from other simulations possible.

Existing LArSoft integration is via exchange files:

- The `celltree` LArSoft module, (Chao).
- Produces a plain ROOT TTree with (hits, digits) and dumps wire geometry (as text).
- Independent from Wire Cell software.
- Not committed, needs a LArSoft package to call home (**suggestions?**).
- Module is not specific to one LArSoft-supported detector.
- Files read in by `wire-cell-sst` package.
- A module to read back Wire Cell output file into to LArSoft is in development.
  - → Need **new charge+point data product** in LArSoft! In mean time, will try to populate closest suitable data products.



Wire Cell / LArsoft Integration By Files

Future Wire Cell / LArSoft Integration

Main technical challenges:

- Both Wire Cell and LArSoft require a large amount of **memory**.
- We expect Wire Cell to be a significant **CPU** bottleneck

Our strategy is to make Wire Cell run in **fine-grained parallel**.

- Parallel at the **time slice** or even the lower "**blob**" level.
- Possible **GPU** acceleration, possible **HPC** utilization.
- Further integration with LArSoft must be understood is in this context.

Options:

- Maybe employ **ATLAS Event Service** approach (actual or DIY)?
  - Need to understand how LArSoft "mothership" communicates to many instances of Wire Cell.
- Enact parallel compute units in the **data flow programming** paradigm?
  - Need parallel execution framework (exists but needs testing)

In any case, let's start by **developing a "`LarWireCell`" module**:

- Wire Cell (core) remains LArSoft "**external**" and needs UPS packaging.
  - Mimic Pandora's integration patterns.
- A LArSoft module needs to convert between data representations and all Wire Cell methods.
- I hope a **LArSoft expert** is interested in helping with this!

Summary

- A working prototype clearly shows the power of the Wire Cell technique.
- A code-refactoring is in progress to provide needed structure going forward.
- Initial samples of 1000s of events have been processed and are available via the Bee online event display.

Future directions:

- **Expert help needed** to improve integration with LArSoft!
- Capture existing **file-based integration methods** into some LArSoft package.
- Explore how to tackle **significant technical challenges** driven by the expected high CPU requirements and in the face of memory pressures from both LArSoft and Wire Cell.

Online Entry Points for More Info

The Wire Cell home page collects all info:

`http://www.phy.bnl.gov/wire-cell/`

Including links to:

- Bee online 3d event display
- Software documentation
- Software repositories