Some Meandering on Experience Using art Tools Redux

Tracy Usher
June 6, 2017

What is an art Tool?

Similar concept to Tools in the Gaudi Framework

- My take: art Tools encapsulate the algorithm paradigm into the art framework
  - Tools provide the basic functionality to perform given tasks for your specific module
  - They are fhicl configurable
    - fhicl to control the list of which Tools to use
    - each Tool can have its own set of fhicl parameters
    - The set of Tools to use and their parameters can be controlled from your job's fhicl control file.

- IMHO, key features:
  - art Tools can be accessed through an abstract interface
    - Allows isolation of specific aspects of a problem to individual objects making code both more readable AND more maintainable (a primary example of code gone bad is SignalShapingServices)
    - Allows several implementations of specific tasks that are configurable in fhicl
  - Can extend this concept…
    - could imagine capability to switch tools "on the fly" based on input information, e.g. processing tracks or showers but through the same interface…

Why an art Tool?

- *Great question!*
  - Already possible to use fhicl to configure standalone algorithms, why do we need to encapsulate our bright shiny algorithms into an art object?
  - In particular, does this make it more difficult to transfer to, say, a Gallery based development environment?
- Primary advantage for larsoft usage:
  - art manages the tools, in particular it handles the object factory
    - If job knows about the tool then it can be instantiated with a simple art macro
    - No need to manage your own object factory
  - Introducing a new tool is a simple process (once you are set up)
    - In particular, if satisfying an abstract interface then switching to the use of a new tool is simply a fhicl parameter change in your job

art Tool Documentation

- Everything you always wanted to know about tools can be found on the art wiki website here.
  - Nice overview with some shell examples to help get started
- Note that art describes “function” tools and “class” tools
  - The former are really simply fhicl configurable functions
    - Carry no state across calls
  - The latter are useful when keeping state (configuration) information across calls
    - Allows fully encapsulated set of functions targeted a specific task
      - e.g. tracking vs shower reconstruction
- I’m focusing on class tools here

An Example

- Let's say you were trying to increase flexibility with the hit finding module:
  - Peak finding stage - allow different algorithms to improve the finding of candidate pulses
  - Peak fitting stage - what can we do to try to improve the fits AND speed up the process?
- One approach is to implement the two tasks above through tool interfaces
  - The Hit Finder module doesn't need to change the way it processes ROI's and outputs hits
  - Can simply change the subtasks with fhicl in the primary job control file

Example Interface (Peak Finding)

```
////////////////////////////////////////////////////////////////////////
///
/// \file   ICandidateHitFinder.h
///
/// \brief  This provides an interface for tools which are tasked with
///         finding candidate hits on input waveforms
///
/// \author T. Usher
///
////////////////////////////////////////////////////////////////////////

#ifndef ICandidateHitFinder_H
#define ICandidateHitFinder_H

#include "fhiclcpp/ParameterSet.h"

namespace reco_tool
{
    class ICandidateHitFinder
    {
    public:
        virtual ~ICandidateHitFinder() noexcept = default;

        // Define standard art tool interface
        virtual void configure(const fhicl::ParameterSet& pset) = 0;

        // Define a structure to contain hits
        using HitCandidate_t = struct HitCandidate
        {
            size_t startTick;
            size_t stopTick;
            size_t maxTick;
            size_t minTick;
            float  maxDerivative;
            float  minDerivative;
            float  hitCenter;
            float  hitSigma;
            float  hitHeight;
        };

        using HitCandidateVec      = std::vector<HitCandidate_t>;
        using MergeHitCandidateVec = std::vector<HitCandidateVec>;

        // Search for candidate hits on the input waveform
        virtual void findHitCandidates(const std::vector<float>&,          // Waveform to analyze
                                       size_t,                             // waveform start tick
                                       double,                             // threshold
                                       HitCandidateVec&) const = 0;        // output candidate hits

        // Search for candidate hits on the input waveform
        virtual void findHitCandidates(std::vector<float>::const_iterator,  // Start of waveform
                                       std::vector<float>::const_iterator,  // end of waveform
                                       size_t,                             // waveform start tick
                                       double,                             // threshold
                                       HitCandidateVec&) const = 0;        // output candidate hits

        virtual void MergeHitCandidates(const std::vector<float>&,
                                        const HitCandidateVec&,
                                        MergeHitCandidateVec&) const = 0;
    };
}

#endif
```

Pretty standard boiler plate to start…
e.g. you should have virtual destructor to keep build system happier

Provide mechanism to configure the class
(note that convention is to supply the fhicl parameter set in the class constructor so not critical to do explicitly)

All your stuff to define the functionality you want to provide

Implentation

- For this task have so far implemented two tools for finding peaks in ROI's:
  - The standard approach which applies a threshold to the waveform
  - An approach based on looking at the (smoothed) derivative
- How to instantiate a given tool in your code:

```
fHitFinderTool  = art::make_tool<reco_tool::ICandidateHitFinder>(p.get<fhicl::ParameterSet>("CandidateHits"));
```

  - where fHitFinderTool is a: std::unique_ptr<reco_tool::ICandidateHitFinder>
- How does art know what tool to instantiate?
  - The interface is the first clue, the specific tool comes from fhicl:

```
candhitfinder_derivative:
{
    tool_type:    CandHitDerivative
    WaveformAlgs: @local::hitfinderwaveformalgs
}
```

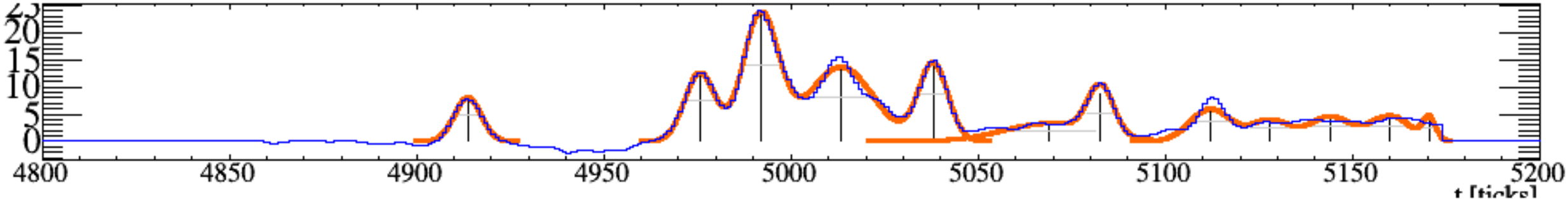
"tool_type" tells art the specific tool

Tools can also instantiate other tools!

Example Output

Note: have modified the even display to draw composite hits (including adding center position for each hit)

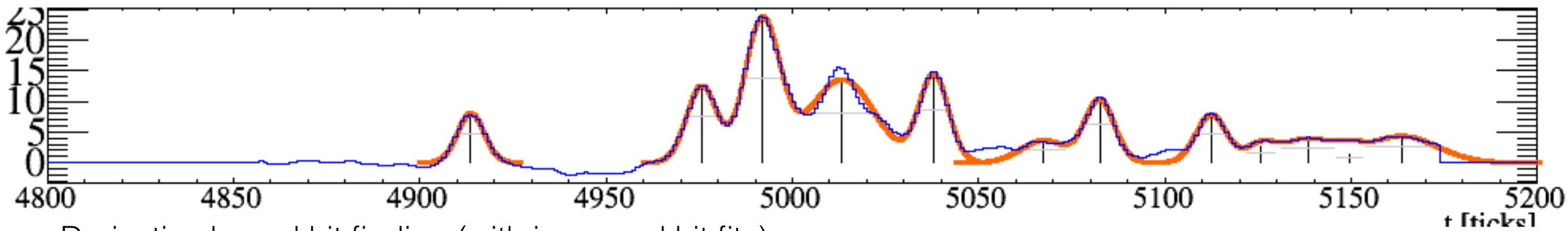
Standard MCC8 hit finder (read from file)



Standard hit finding (but have improved hit fits)

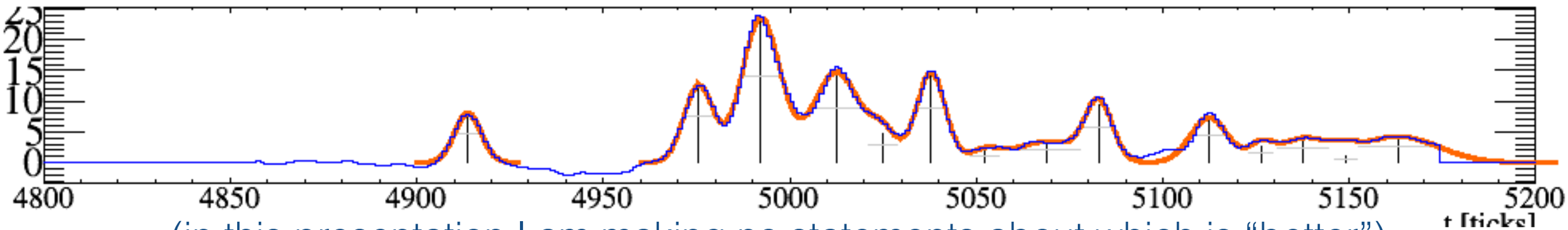
physics.producers.gaushit.CandidateHits: @local::candhitfinder_standard

← fhicl override in top level job file



Derivative based hit finding (with improved hit fits)

physics.producers.gaushit.CandidateHits: @local::candhitfinder_derivative

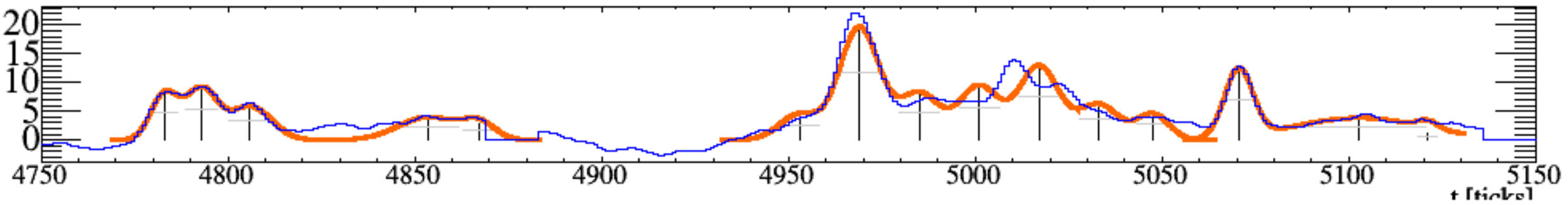


(in this presentation I am making no statements about which is “better”)

Example Output

Note: have modified the even display to draw composite hits (including adding center position for each hit)

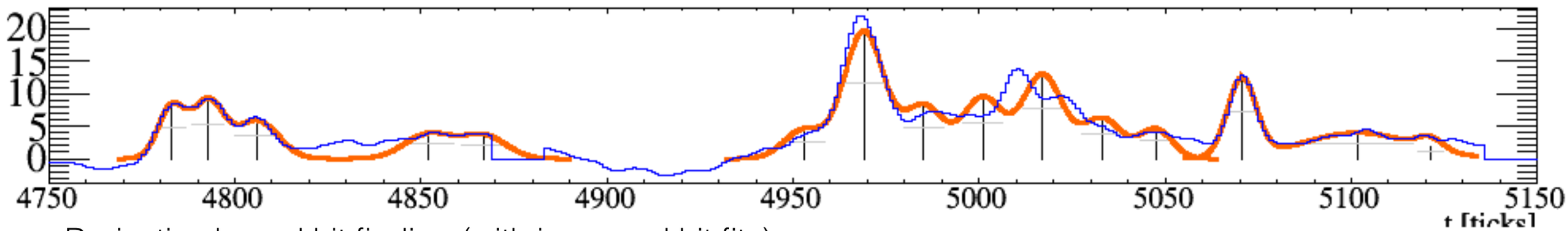
Standard MCC8 hit finder (read from file)



Standard hit finding (but have improved hit fits)

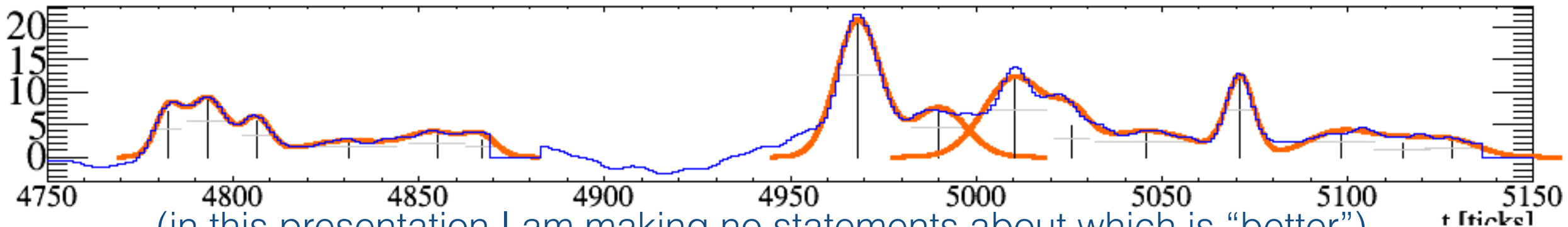
physics.producers.gaushit.CandidateHits: @local::candhitfinder_standard

fhicl override in top level job file



Derivative based hit finding (with improved hit fits)

physics.producers.gaushit.CandidateHits: @local::candhitfinder_derivative



(in this presentation I am making no statements about which is “better”)

Other Recent Tools Examples

- MicroBooNE:
  - Have revamped “CalWireROI”
    - Responsible for deconvolving raw waveforms
    - Determines “Regions of Interest” on the wires
  - Tasks broken into tools:
    - Can switch techniques for identifying ROI’s
    - Can switch between “full wire” and “ROI” deconvolution
- ICARUS:
  - Generally, have been following the MicroBooNE model
    - But MicroBooNE’s version of SignalShapingServices is, well, opaque
  - Have rewritten Signal Shaping Services for use in ICARUS with tools
    - Field response - currently 2 implementations
    - Electronics response - currently only one implementation
    - noise - currently 3 implementations

Wrap Up

- Tools are a new concept in art
  - introduced with larsoft v06_26_00
- The concept provides a mechanism to manage algorithms which implement specific functionality for your modules
  - art handles the tool management
  - tools are fhicl configurable, each tool implementation can have its own set of fhicl parameters
- Regardless of whether you embrace the art tool model explicitly, we need to make a conscious effort to move to a more algorithm based approach in our code
  - Otherwise we are going to face real maintenance nightmares going forward as the first generation of code writers move on to other projects!