LArLite

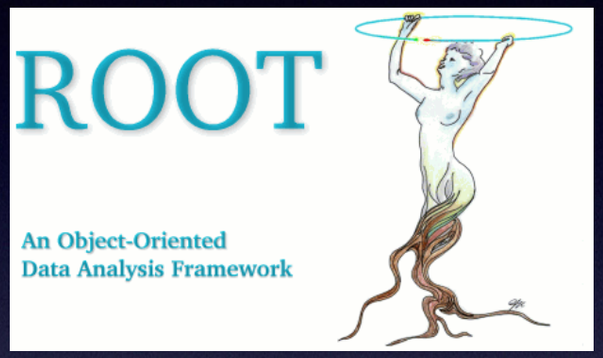
Introduction & Use Case

Kazuhiro Terao @ Nevis, Columbia

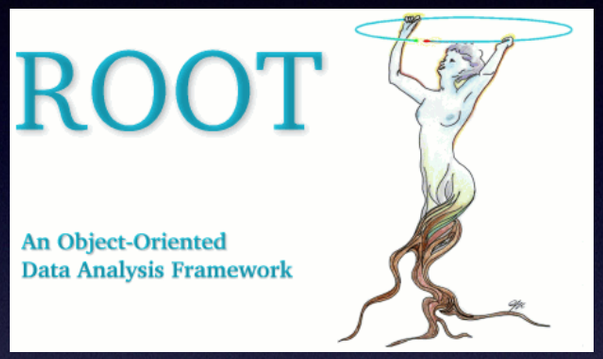
About LArLite

What Is LArLite?

- C++ code development toolkit
  - Goal = easy & simple C++ code development
  - "Supports Darwin & Linux"
  - Dependency: ROOT, git, & LLVM or GCC
    - ROOT for Cling/CINT dictionary generation



What Is LArLite?



- C++ code development toolkit
  - Goal = easy & simple C++ code development
  - "Supports Darwin & Linux"
  - Dependency: ROOT, git, & LLVM or GCC
    - ROOT for Cling/CINT dictionary generation
- Typical code development steps

1. Generate a repository (a unit for collection of "package" directories)
   - Can be a user's git repository … typically github is used
2. Generate a package in a repository (a unit for library generation)
3. Write/Compile code in a package
4. Use compiled library
   - Class/Function access through Cling/CINT/Python
     - Python compat. largely through PyROOT, implementing option to build using Cython
   - … or compile an executable
   - Write an extension (separate) package and link against

“Easiness”

- Easy to install
  - Many has ROOT, git, & compiler to build ROOT on their machine
  - … then all needed is to “git clone” from github

“Easiness”

- Easy to install
  - Many has ROOT, git, & compiler to build ROOT on their machine
  - … then all needed is to “git clone” from github
- Easy to write code
  - Repository, package, C++ class empty source code generation scripts
    - One owns everything generated if started from scratch
    - No need to “parasite” existing code repository
    - No need to “copy & paste” existing source code
  - “Fast” compilation (i.e. only compile what you wrote)

“Easiness”

- Easy to install
  - Many has ROOT, git, & compiler to build ROOT on their machine
  - … then all needed is to “git clone” from github
- Easy to write code
  - Repository, package, C++ class empty source code generation scripts
    - One owns everything generated if started from scratch
    - No need to “parasite” existing code repository
    - No need to “copy & paste” existing source code
  - “Fast” compilation (i.e. only compile what you wrote)
- Easy to use compiled code
  - Cling/CINT/Python interpreter to immediately access class/functions

```
Kazus-MacBook-Pro:MyExample kazuhiro$ root -l
root [0] sample()
(class sample)140407275687440
```

Accessing “sample” class
in CINT

```
>>> from ROOT import sample
>>> sample
<class 'ROOT.sample'>
```

Accessing “sample” class
in Python

“Easiness”

- Easy to install
  - Many has ROOT, git, & compiler to build ROOT on their machine
  - … then all needed is to “git clone” from github
- Easy to write code
  - Repository, package, C++ class empty source code generation scripts
    - One owns everything generated if started from scratch
    - No need to “parasite” existing code repository
    - No need to “copy & paste” existing source code
  - “Fast” compilation (i.e. only compile what you wrote)
- Easy to use compiled code
  - Cling/CINT/Python interpreter to immediately access class/functions
- Easy to share code
  - User A can simply git-pull user B’s repo through github & compile/use
  - … or if it’s popular can leave in larlite repository

What Users Liked About

- Very easy to get started. Simple to understand.
- Well documented.
- I don’t have to start code-writing by “copy & paste” nor “build with 20 already-existing .cxx code.”
- Easy to develop my own code suit or even framework.
- I graduated from CINT! I have my own compiled toolkit!
- Fast compilation to just compile my code
- My code in my git repo. Easy to share w/ others.
- My Python code is now “fast” with compiled C++ library
- I can use bunch of python apps with my C++ code now!

LArLite Needs?

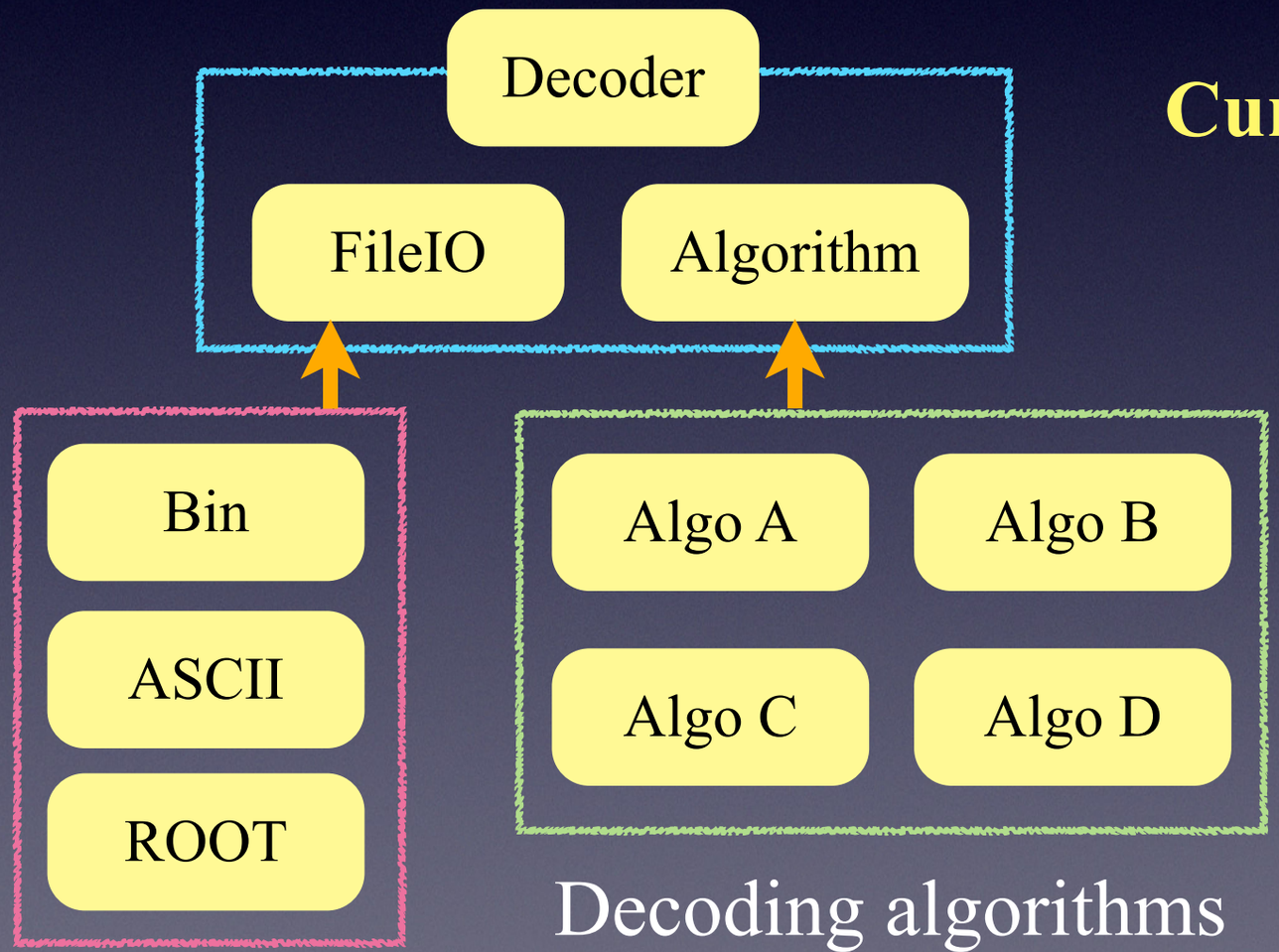
- "Light-weight" installation
  - small set of dependencies: ROOT, git, and llvm/gcc
- Simple "start from scratch"
  - write an independent code suit in his/her own git repository
- CINT/Cling dictionary generation support
  - compiled C++ code available in C/Python interpreter
  - try compiled class/function w/ an interpreter immediately
- Flexibility
  - smaller group, minimal policies, simple design
- Portability
  - obviously "standard C++ source code" can be exported outside

Use Case Examples

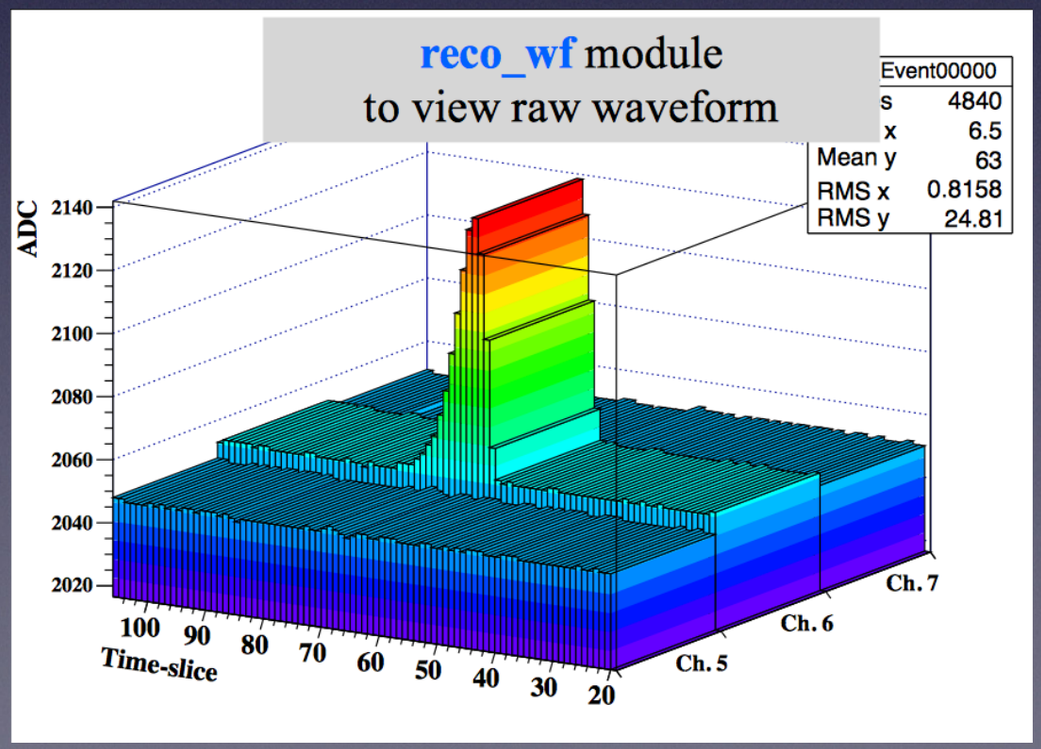
Use Case Example

Binary Decoder

- 1st project: store an waveform as C++ data product in ROOT file
  - More intuitive access for students familiar w/ C++ and ROOT
- Have a simple framework to interface various binary format



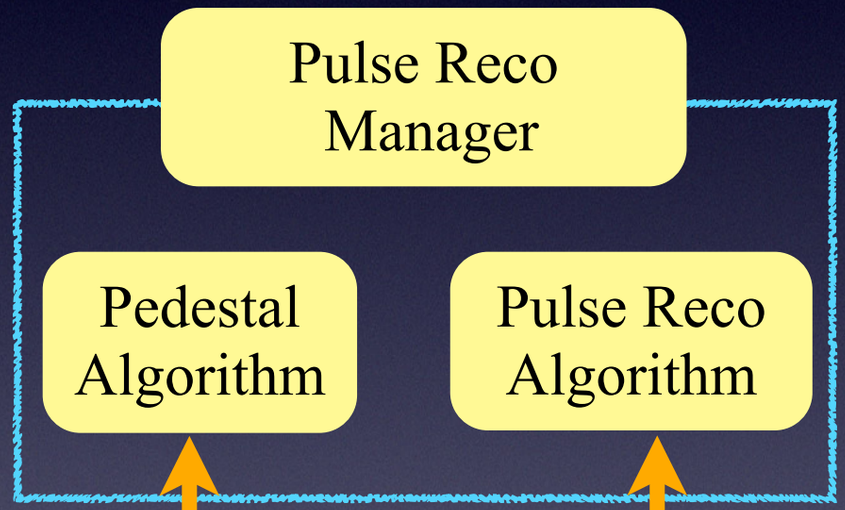
Current: low level decoder used for FPGA debugging

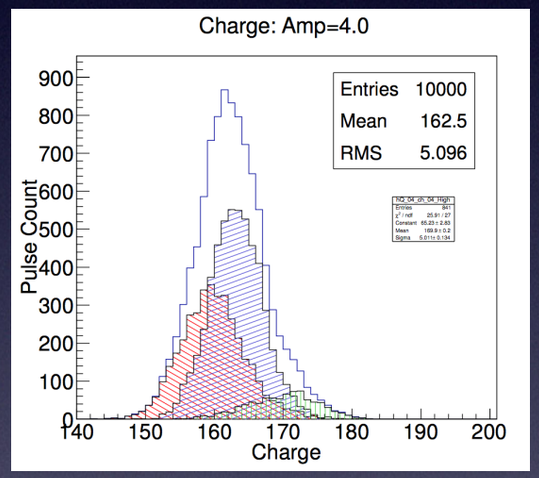


Use Case Example

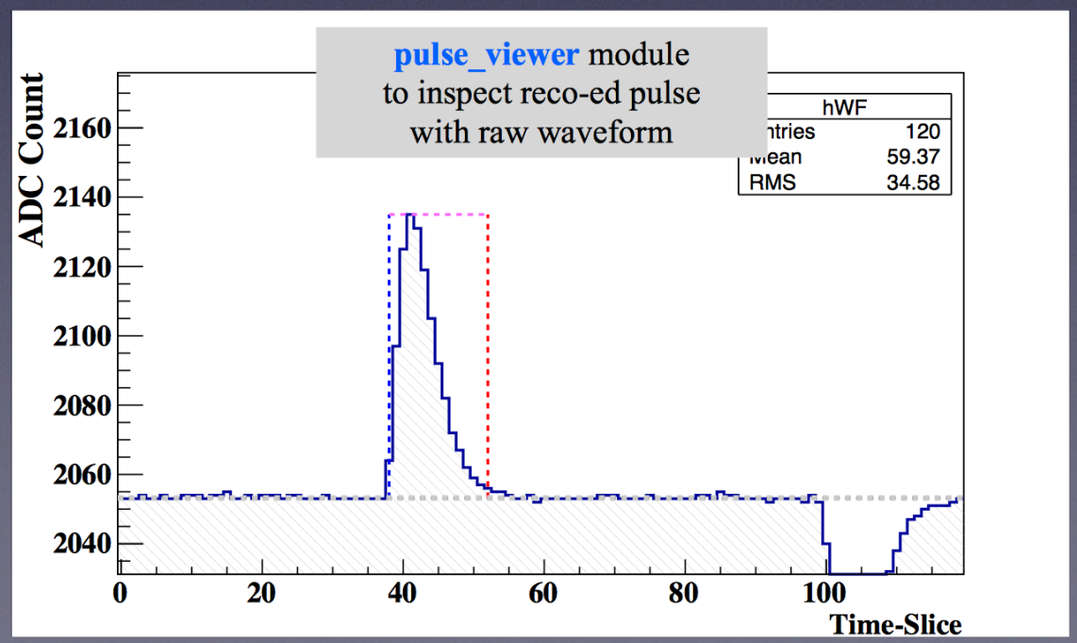
Optical Pulse Reconstruction

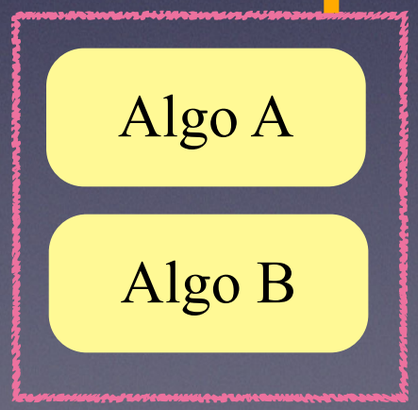
- A summer undergraduate student liked C++ and wanted to practice
- He made a similar framework for optical pulse reconstruction



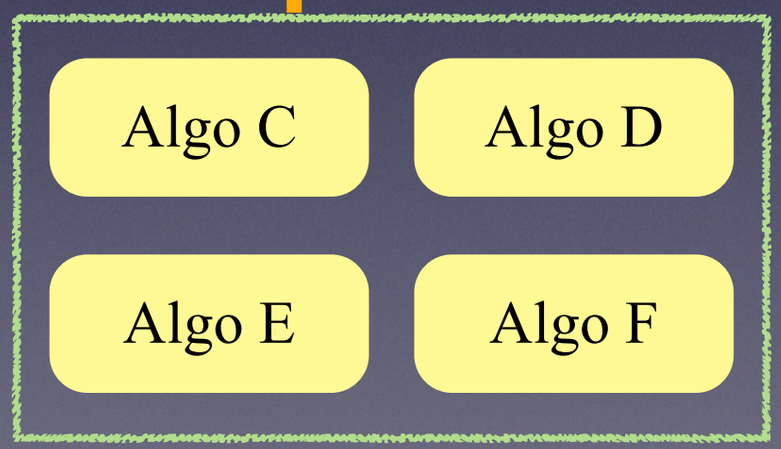


Current
Used for optical pulse reco for MicroBooNE





Pedestal Algorithms

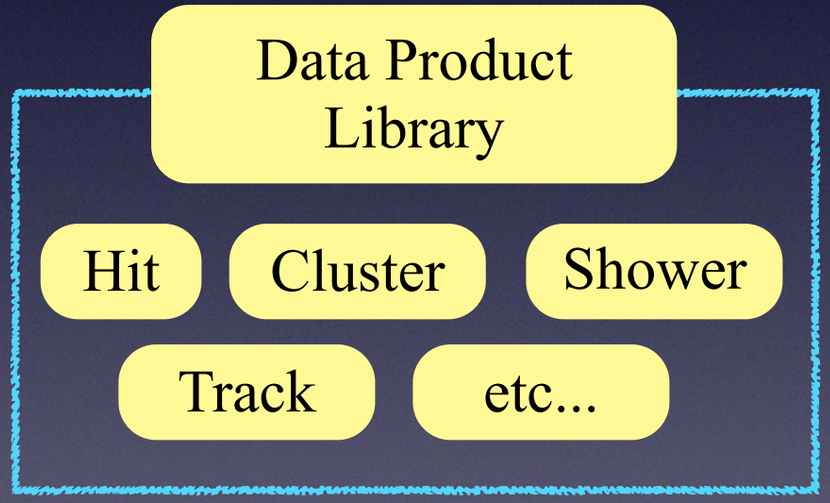


Pulse Reco Algorithms

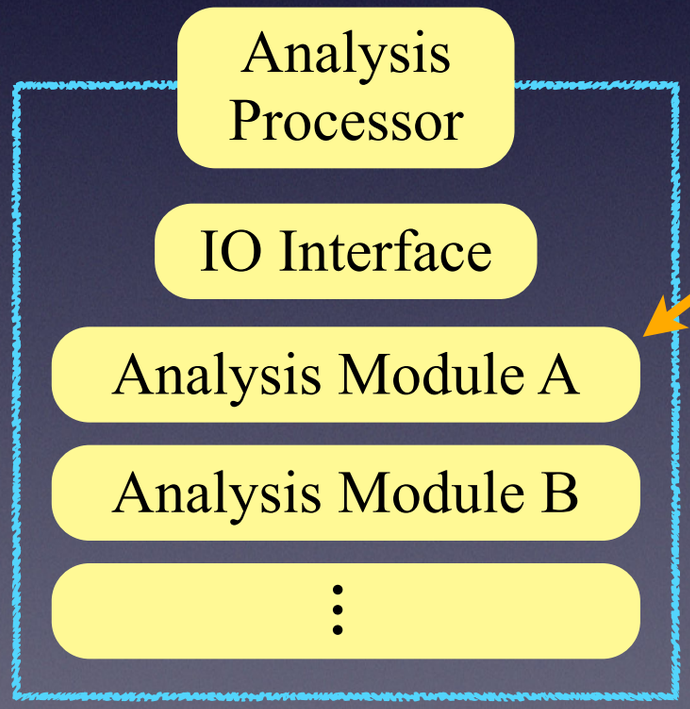
Use Case Example

ala LArSoft Analysis

- A graduate student was suffering from using LArSoft
- He came and ask if somehow this can be faster and run on his laptop
- Made analysis framework with identical data product def. as LArSoft



ala LArSoft DataProducts



Processor Fmwk (from pulse reco)

User's module

Current

Used for many analysis/reconstruction in LArLite

Use Case Example

Optical Detector Simulation

- Needed C++ simulation for our optical readout electronics
- Wrote a suite of simulation chain

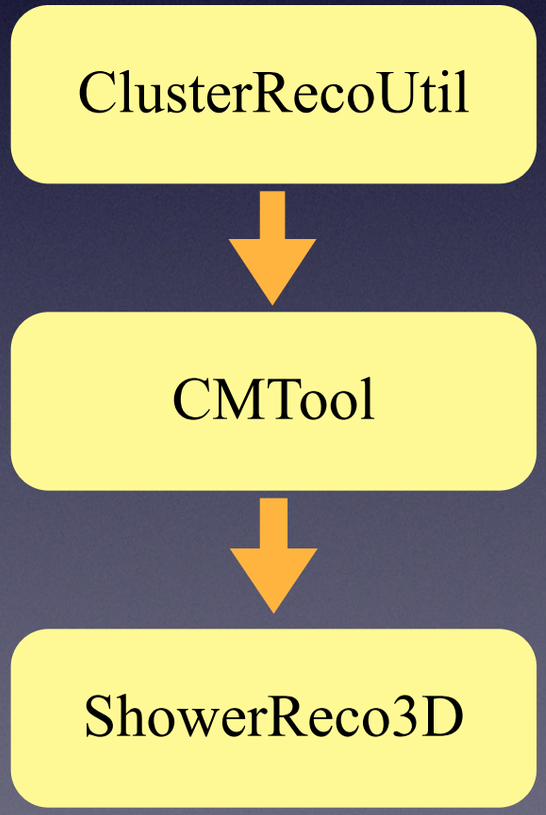
ADC Simulation	Modulated algorithm to simulate optical detector pulse shape
FEM Logic (FPGA) Simulation	FPGA logic in C++
Trigger Logic (FPGA) Simulation	FPGA logic in C++
DRAM Readout Simulation	Producer of “raw” waveform

Current
Ported into LArSoft for UB optical readout simulation

Use Case Example

EM Shower Reconstruction

- Other students/post-docs tried LArLite and liked it (faster, easier)
- The majority of users are interested in making EMShower reco
- Wrote a suite of reconstruction chain for EMShower



Detailed 2D cluster parameter algorithm & data representation

2D Cluster Merging/Matching Framework

3D EM shower reconstruction Framework

Current
Ported into LArSoft for EM Shower reco

Use Case Example

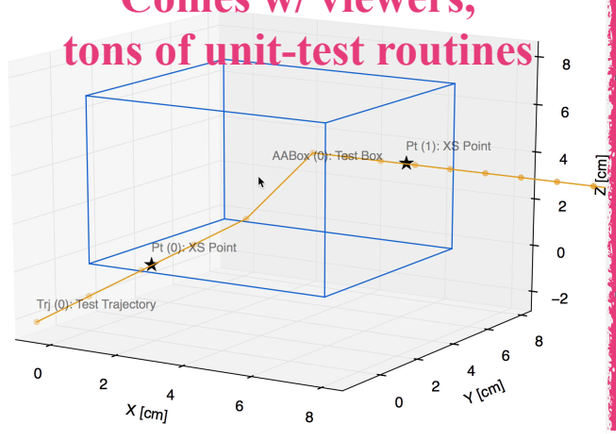
GeoAlgo

- Wanted a suite of geometrical calculation algorithms for analysis/reco
- Decided to make our own based on a text book

"The Book": Real Time Collision Detection

- Algorithm/Design pattern for 3D game programming
- Highly regarded reference among experts
- Excellent resource for our problem solving!
- Easy to read & follow
  - Lots of abstract code template in the book

Comes w/ viewers, tons of unit-test routines



Current

Used in various analysis!

Use Case Example

LArPy

- A converter suite from C++ objects to Python built-in types
  - This is for toolkits introduced in LArLite
  - Uses Python native C-API, no extra dependency
- Helps to bridge with Python open source applications
  - e.g.) current GeoAlgo viewer uses this suit + matplotlib python module
- On-going work but effort shifting to larlite_numpy

larlite_numpy

- On-going work!
- Following root_numpy approach
- Uses “ala LArSoft” data libraries to create numpy record array
  - Using Cython (… and with very small Python C-API)
- Many awesome Python scientific libraries available for analysis
  - num/scipy, pandas, PyTable, scikit-learn, PyQTGraph, matplotlib, etc…

Use Case Example

Understanding LArSoft

- Compared a speed performance of a simple analysis
  - Read in data product, make a TH1D histogram
  - Resulted in ~5 orders of magnitude difference in speed
- Identified a major cause in our usage of art (at least in MicroBooNE)
  - We had a “service” (singleton) that was always doing heavy analysis
  - It was always there, no one noticed, often not used.
  - LArSoft users were used to “~1 second / event” process speed
- Identified a need of improvement in art utility
  - After fixing the problem above, still slow
  - FindManyP in LArSoft was slower than an equivalent suite in LArLite
  - Feedback to art team by Wes Ketchum, now LArSoft is pretty fast

Use Case Example

Well, there are more & more of applications made in LArLite

Sensitivity Study

Optical Model Analysis

DataCompression Modeling

Event Reconstruction Framework

DataProcessing Framework

RunConfiguration Database Tools

LowEnergyExcess Study

Calibration DB Interface

What I am getting from this experience:

- Providing a support greatly help to speed up code development
  - Most code written by students, and they do enjoy a lot
  - One undergrad student could write a fmwk with I/O by herself
- ... and good news: there are undergrads, grads, and post-docs who really want to write a proper code suite rather than a CINT macro :)

Summary

Summary

- LArLite is a simple & light-weight code development suite
  - Easy to install
  - Easy to develop/extend code, build, and share with others
  - Easy to write an application w/ interpreter support
- Many code toolkits/frameworks written using LArLite
  - This keeps on-going…
  - Includes “ala LArSoft” analysis framework
  - Large fraction exported to LArSoft
    - Raised questions on how to share/maintain code written outside LArSoft