FNAL Software School: Lecture 1
Course Code Infrastructure
Cluster and Hit Reconstruction

Matt Herndon,

University of Wisconsin – Madison

Course Code Base

- Location
  - https://github.com/herndon/FNALComp/tree/production
- Organization
  - Framework: Event processing framework
  - Modules: Produce or consume data objects each event
  - DataObjects: Class definitions of basic data objects
  - Geometry: Definition of the detector geometry and properties
  - Algorithms: Algorithms that run on the data objects to perform operations like IO, fitting, …
  - Tracking: High level tracking classes with algorithms and supporting functions
  - Services: Services such as Errorlogger, Random number generation …

Code Structure

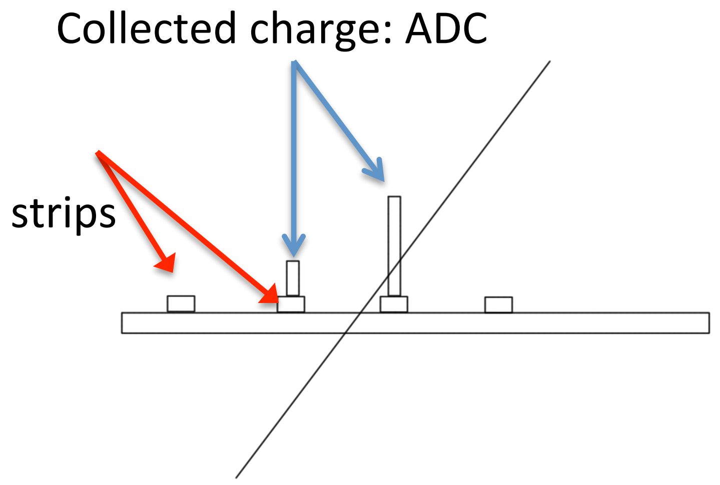
- Main program and Event processor: dataRead.cc
  - Initializes persistent objects used throughout the program
  - EventProcesser: process “Events”, runs the processEvent function of Modules once per event
  - Other persistent objects: Config, DetectorGeoemtry, Random, Iostreams, Root file… persistent objects over the scope of the whole program
- The Event: Event
  - Particle physics takes place in short time scale interactions or “Events”
  - This informs us on how to structure our code
  - Event: Contains DataObjects persistent for the scope of one Event
  - Modules can read (consume) and/or write (produce) data objects from/to the event
- Modules: HitRecoModule
  - A class that performs a specific task producing or consuming (frequently both in reconstruction) data objects
  - Specific instances of Modules are derived classes inherited from the Module class to give a common class type and interface that EventProcessor can control.
  - Special cases are CountEventsSource, starts generation of data, and DataSource, reads data from disk.

Code Developers

- Your role
  - As a code developer performing reconstruction:
  - Develop Modules for performing a single specific task. Today you are using the premade module template HitRecoModule
  - Develop algorithms to process data within a Module. Today you are performing cluster finding and hit reconstruction in HitRecoModule.
  - You will consume as input a StripSet and produce as output a HitSet

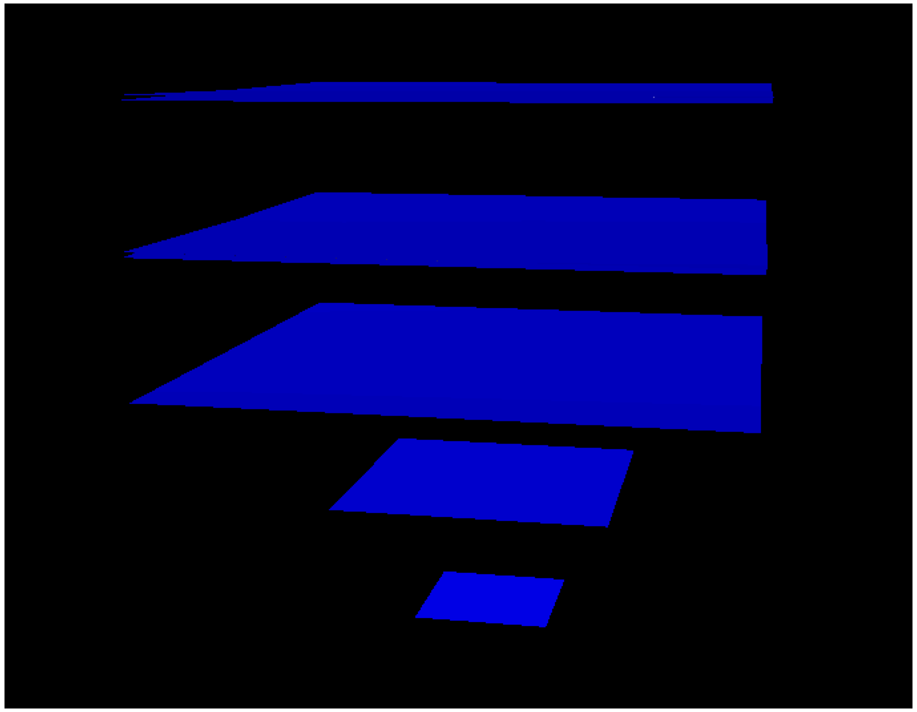
Cluster and Hit Finding

- Cluster and hit finding
  - Goal: Efficiently find the estimated position where a charged track traversed the detector
- Strip detector
  - Silicon strip detector, calorimeter strip chambers, muon detector.
  - In our case charge is deposited relatively on the adjacent strips in inverse proportion to the distance to the strips.
  - Location found from charge weighted sum



- Final result, Hit: 1D measurement with known position (and resolution)

Strip Detector



- 10 Layer of strip planes arranged in 5 closely spaced pairs (200um)
  - Three pairs with X and Z measurements
  - 2 pairs with X and SAS (small angle stereo, 1^o rotated from X)
  - Intersection of X and SAS strips allows effective measurement in Z
- Sufficient determine helical trajectories of a track in magnetic field.

Strip Detector

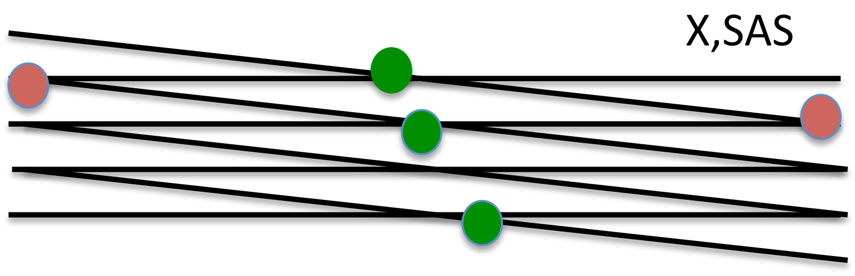
- Detailed properties
  - Described in sensonsorgemetry.txt

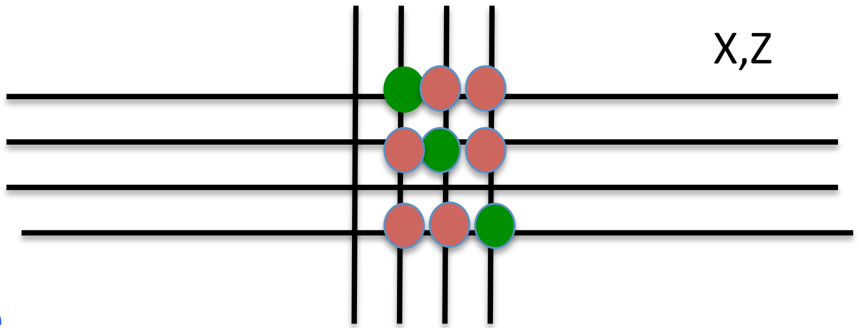
TABLE I: Sensor properties.

Layer	type	Number Strips	Strip Pitch (um)	Y Pos (m)	Res (um)
0	X	2048	50	0.2	12
1	X	2048	100	0.4	25
2	X	2048	200	0.6	50
3	X	2048	200	0.8	50
4	X	2048	200	1.0	50
5	Z	2048	50	0.2002	12
6	Z	2048	100	0.4002	25
7	Z	2048	200	0.6002	50
8	SAS	2048	200	0.8002	50
9	SAS	2048	200	1.0002	50

Strip Detector

- Why SAS?
  - Intersection of X and SAS strips gives effective measurement in Z
- Ambiguity
  - Less ambiguity from SAS layers.
  - If you had only SAS (or worse Z) layer the same pattern would repeat at every layer and all the false pairings would look real!
  - Different types of layers breaks the ambiguity.
- Why Z:?
  - More accurate Z measurements
- Angular resolution
  - This detector has the good angular resolution in the outer layers
  - Makes sense to start tracking there



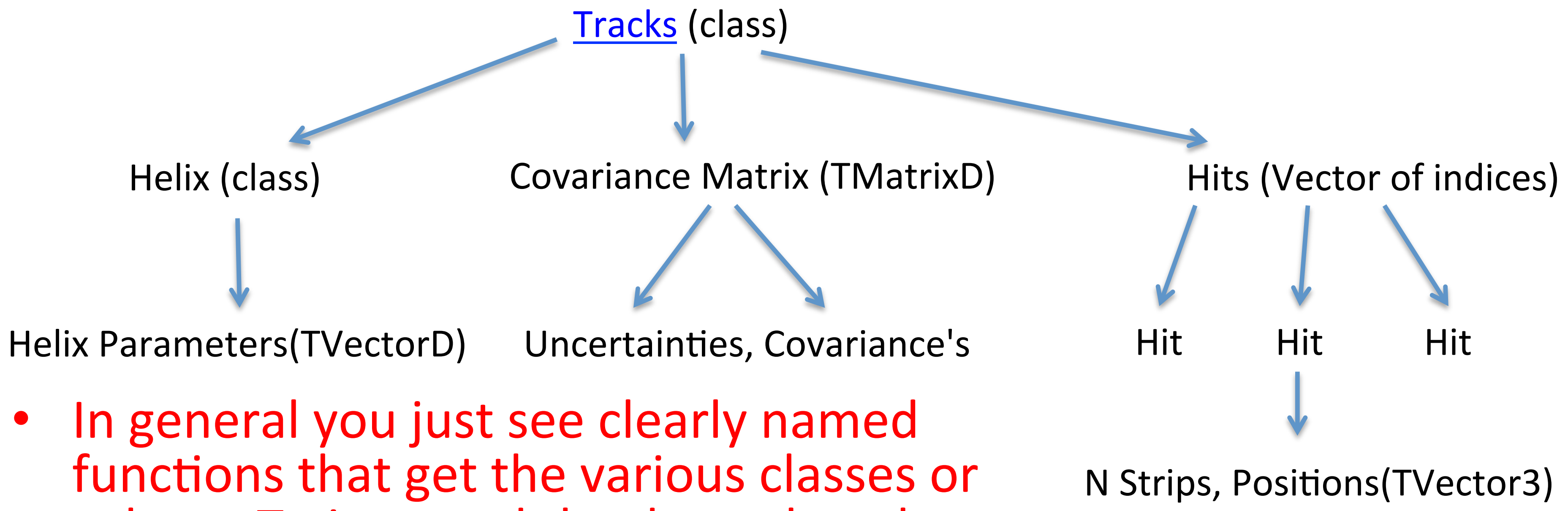


Programming Project

- Cluster and Hit Finding applying algorithm abstraction
- Project goal summary
  - Given raw data of strip number and ADC counts from a strip detector determine location where the tracks traversed the detector layers.
- Two primary logical steps
  - Cluster finding: Find all adjacent sets of above threshold strips that may be due to one or more tracks traversing the sensor at that location.
  - Hit construction: Convert that information into a location in a global coordinate system
- Software engineering concept:
  - Algorithm abstraction: Determine the logical steps of your problem and map them to abstractions
  - The abstractions are the names of the functions that will accomplish each step.
  - Our problem, though not complex, is complicated enough that we will create a hierarchy of abstractions to represent our problem.

Abstractions

- The DataObjects in our code represent a hierarchy of abstractions as well.


Tracks (class)
Helix (class)
Covariance Matrix (TMatrixD)
Hits (Vector of indices)
Helix Parameters(TVectorD)
Uncertainties, Covariance's
Hit
Hit
Hit
N Strips, Positions(TVector3)

- In general you just see clearly named functions that get the various classes or values. To just read the data what classes and types of containers hold the data should be largely irrelevant and can be abstracted away from the user.

You as the Developer

- Your job as the code developer
- Develop the abstractions
- Your cluster finding code should read like
  - ReconstructsHits
  - (loop over layers)
    - findClustersOnLayer
    - (Loop over clusters)
      - …
- Note there are often multiple reasonable algorithm choices
  - Find all clusters on layer. Process all clusters on a layer to make Hits
    - For each Layer
    - Create a vector of all the cluster acd information where the cluster acd information consists of multiple strips and can also be represented by a vector. A vector of vectors. Create a second vector of initialStrips
    - Or you could make a vector of indices, strip numbers, into the StripSet representing the start of each cluster.
  - Find one cluster. Process that cluster to make a Hit. Look for the next cluster.
    - Have to keep track of position in the StripSet
    - I did this (we should compare!)

Me as the DataObject Developer

- Also make optimal decisions about what underlies the abstractions, Consider the StripSet
  - StripSet: vectors of maps (key int strip number, value int acd)
  - Data is sparse. In the map not every strip number needs to be there
  - Container is ordered, indexed by key and keys are unique.
    - When filling we can insert strips into the order. If the key already exists we should add the additional ACD counts.
    - The strips we insert may go anywhere since they are at the locations of track intersection. Maps support fast insertion into the map order.
    - We can insert or retrieve a strip ADC value by stripNumber index.
    - The ordering is needed for cluster finding
    - Note the insertStrip function is an abstraction. I can(should now!) change it whenever I want without effecting the user of the interface.

Look for help

- Cluster and hit finding is the inverse of what was done to generate the data.
  - There we found the position where the tracks intersected layers and converting those hits to clusters of strips
- HitStripGenModule
  - Look at storeStripInfo
    - calclateLocalFromGlobalPostion
    - calculateStripFromLocalPosition
    - generateClusterFromStripHitPosition
- The calculate functions are free functions.
  - Looking at the file with the functions we see the inverse calculation functions have already been written!

Project Specifications

- Input: StripSet, raw data from the strip detector
  - vector of 10 maps with pairs of key int strip,value int acd information
    - Access: retrieve each map and iterate over the elements
- Output
  - Intermediate: clusters per layer: vector of vectors of acd values, vector of initial strip numbers
  - Intermediate: Hits with TVector3 position, number of strips, charge
  - Final: HitSet, vector of Hits
- Given the specifications there is only one correct solution to the HitReconstuction problem
  - After adding a print method similar to those for the other DataObjects the output should be identical to the “make test” output

Project Specifications

- Algorithm
  - reconstructsHits
  - (loop over layers)
    - findClustersOnLayer
      - (loop over strips)
        - » test if adjacent
          - add to single cluster adc vector if adjacent
          - add single cluster adc vector to vector of clusters if not adjacent
      - (end loop over strips)
    - (Loop over clusters)
      - buildHit
        - » calculateStripHitPositionFromCluster
        - » … coordinate transformations
        - » make the Hit
      - storeHitInSet
    - (end loop over clusters)
  - (end loop over layers)

Course Goal Revisited

- Goal
  - Learn how to write well designed and effective reconstruction software that integrates well into a large scale computing project
- What does that mean?
  - Follows best practices
    - Many of the best practices are there to facilitate the elements of the goal.
  - Easy to read
    - A user or other developer can read and understand quickly what your code does.
  - Easy to maintain
    - Need to improve something? Well designed code will often let you do so with a change at a single point without effecting any of the classes and functions that use the code you've changed.
  - Simple
    - The simplest solution is used when various solutions are equally effective.
  - Safe
    - Data elements are safe from being altered when they should not be.
  - Fast uses minimal memory
    - A fact of particle physics computing is that we deal with large data sets and are CPU and memory limited.
  - Effective
    - Defined in terms of the project goal. In reconstruction typically, efficient, accurate, and low fake rate (reconstruction of Hits, Tracks that don't exist!) reconstruction of objects.

Algorithm Abstraction