DUNE Doc-xxxxx

TPC Simulation and Track Reconstruction in GArSoft

Leo Bellantoni[a], Eldwan Brianne[b], Thomas Campbell[c], and Thomas Junk[a]

[a]*Fermi National Accelerator Laboratory,* [b]*DESY Hamburg,*
and [c]*University of Colorado at Boulder*
April 9, 2019

This document describes the simulation and reconstruction tools available in the `GArSoft` software package for TPC tracking. This note focuses on tracking simulation and reconstruction – ECAL software is described elsewhere. These tools include event generation, particle interaction simulation and energy deposits, drift and diffusion modeling, field response and electronics response, digitizaton, hit finding and clustering, track finding and fitting, and vertex identification. Details of the algorithms and the associated data representation are given. The performance of the algorithms using GENIE neutrino interactions is summarized.

I. SOFTWARE ENVIRONMENT

`GArSoft` was created in 2017 by Brian Rebel using copies of tools developed for the `LArSoft` toolkit [1]. There is no linkage to LArSoft libraries or code at the moment, though `GArSoft` shares dependencies on many of the same external products. Some of the important external dependencies are *art*, the event-processing framework [2], `NuTools`, `ROOT`, `GEANT4`, and `GENIE`. `GArSoft` source code is stored in Fermilab's Redmine system using git for version control [3]. `GArSoft` is compiled and linked using `mrb`, the multi-repository build system [4]. The build system makes a `UPS` product [5] that, when set up using UPS, sets up all of the external dependencies. Our current development environment assumes that the proper external dependencies are available to be set up via `UPS`. Setting up the DUNE software environment via

```
source /cvmfs/dune.opensciencegrid.org/products/dune/setup_dune.sh
```

is sufficient to set up `UPS` and to define the `PRODUCTS` environment variable which is a search list of directories in which `UPS`'s `setup` command looks for products that are needed for a full setup. Currently it is most convenient to use pre-built external products that are installed in `CVMFS` [6] as set up by the above script. `CVMFS` on macOS is known to be very slow. We currently recommend developing on Linux, either with `SL6` or `SL7`-compatible variants, though macOS builds have been performed in the past.

Code is expected to compile under `gcc` and `clang`. Developers normally build and test with just one of these, but periodically maintainers build with the other. If a bug is found by compiling with the other compiler, we have been informal about it – if the fix is easy, such as commenting out the declaration of an unused variable, the fix can simply be checked in to the repository. More involved issues need to be forwarded to the original author of the code. The version control system `git` has a `git blame` feature which is quite useful, and the Redmine web interface to the repository also provides an "annotated" view which highlights each line of code and refers to the person who committed it and provides a link to the commit.

The `GArSoft` redmine wiki page [3] includes instructions for setting up the software environment, downloading the code, compiling it, running test cases and viewing the results using the event display. Instructions are also provided for producing an analysis ntuple rootfile that can be analyzed with ROOT without modifications or additions.

While it is still early in the software development, plans are made so that `GArSoft` and `LArSoft` can interoperate. Because of the upstream Pixel LAr detector and the need to simulate events that produce ionization signals in both the liquid and the gas, we should not preclude the possibility of running `LArSoft` modules in the same *art* jobs as `GArSoft` modules. For this reason, the namespace `gar::` is being used extensively in `GArSoft` in order to prevent name clashes. Not everything's perfect, and we may have to rename some things if we ever set up and run `LArSoft` and `GArSoft` together, as *art* also relies on the filenames of modules and services to identify them.

II. EVENT GENERATORS

Five event generators are provided. Each one produces `simb::MCTruth` data products, which are defined in `NuTools`. The particle gun generator is in `EventGenerator/SingleGen_module.cc` and is configured with `prodsignle.fcl`, with shared parameter definitions in SingleGen.fcl. It can produce one or several particles at fixed energies and angles, or with a distribution of energies, angles, and particle species. It is modeled after LArSoft's particle gun generator.

A text-file generator is also provided, EventGenerator/TextFileGen_module.cc, which is configured with prodtext.fcl. It inputs the particle information from a text file and copies it into simb::MCTruth data products for each event. One must be sure to use the -n nevents option on the command line where nevents is less than or equal to the number of events in the text file, or the generator will throw an exception.

GENIE, currently at version v2_12_10c, is a neutrino-nucleus interaction generator. GArSoft includes an interface to NuTools's GENIEHelper in the module EventGenerator/GENIE/GENIEGen_module.cc, which is configured by the top-level prodgenie.fcl as an example, and shared configuration is available in GENIEGen.fcl.

A Radiological generator is also provided, copied from the one in LArSoft. Its code is in RadioGen_module.fcl. Currently it has not yet been tested. The LArSoft version reads files containing decay spectra in larsoft_data which is not set up by default in GArSoft.

A cosmic-ray generator CRY is also provided but not yet tested.

Event generators' interfaces to the random number services were upgraded for compatibility with *art* v3, but the seed configuration has not been estensively tested. The default generator fcl configurations generate the same events every time they are run which is useful for debugging during software development. It is less useful for generating large samples in grid jobs, however.

III. PARTICLE INTERACTION SIMULATION AND ENERGY DEPOSIT MODELING

A module, GArG4/GArG4_module.cc, is patterned on the similar LArG4_module.cc in LArSoft, which searches the input event for all simb::MCTruth data products and runs a GEANT4 simulation using the detector geometry. The output data products are of the types simb::MCParticle, gar::sdb::EnergyDeposit and gar::sdp::Calodepost. A constant magnetic field of 0.4 T pointing in the x direction, which is parallel to the electric field, is assumed.

Event generation and the GEANT4 step are currently run in the same jobs but it can be split up if need be.

IV. DRIFT, DIFFUSION, FIELD RESPONSE AND ELECTRONICS RESPONSE

The energy deposits simulated by GEANT4 are then read in by the ReadoutSimulation/IonizationReadout_module.cc which produces data of type gar::raw::RawDigit. The ionization and scintillation is modeled using an untuned Birks' model for recombination. Currently scintillation photons are not simulated in the gas. Electron drift is simulated by numerically integrating over the longitudinal and transverse diffusion distributions, applying a constant drift velocity, and an exponential electron lifetime factor. Electrons are registered on the nearest readout pads after the diffusion simulation has displaced them by a random amount proportional to the square root of the drift time.

Effects of charge induced on nearby pads are not yet included but are expected to have a positive effect on the point resolution of hit clusters. Pulse shaping by the readout electronics is also not yet included. The signal gain is arbitrary at the moment. No noise is included in the simulation yet.

V. DIGITIZATION AND RAW DATA REPRESENTATION

The ADC sampling period is assumed to be 198 ns, which is adjustable in DetectorInfo/DetectorClocks.fcl. An event readout is defined as 18048 time samples.

Raw waveforms are stored with zero suppression using a simple thresholding technique. In addition to all of the ADC samples over threshold, five ticks before the first above-threshold sample are saved, and five ticks after. The threshold and early- and late- tick counts are adjustable in ReadoutSimulation/TPCReadoutSimAlg.fcl.

The output is of the form RawDataProducts/RawDigit.h. The zero-suppression algorithm, which is implemented in RawDataProducts/raw.cxx packs block index information into the ADC vector. This includes a table of contents – a number of blocks over threshold, their starting tick indices, and their lengths. After the table of contents, the blocks are listed. ADC values are stored as signed shorts. The number of samples is an unsigned short. This latter may need to be expanded as it has proven to be a limitation in LArTPC's seeking to store long events. The main purpose of analyzing long events however is to study low-frequency noise. The size of the ADC vector retains the full information about the size of the data even if the too-short sample count has overflowed. The issue arises because of the size of a buffer needed to store un-zero-suppressed data which needs to be specified. If the data are not zero suppressed, which may be the case for special low-frequency noise runs with long events, then the size of the ADC vector is sufficient. If a long event has been zero suppressed, a pass through it may be needed to determine how much memory is needed to store the zero-unsuppressed data.

For now, all data are assumed zero suppressed and event tick counts are assumed to fit in an unsigned short. As it is, the data are never unpacked into a format with the zeros replaced in order to save memory and CPU time, however. Given the large channel counts and the sparse nature of the data, it is encouraged never to store continuous waveforms for the entire detector for an entire event in memory but rather to work with the zero-suppressed blocks.

VI. HIT FINDING AND CLUSTERING

The first stage of a reconstruction job is to find hits from the raw digits. ALICE uses the word “hit” to mean Monte Carlo charge depositions, but `LArSoft` uses the word “hit” to refer to a reconstructed object – a pulse that has been found and fit on a single channel. We follow the latter convention.

Hits largely map onto the over-threshold blocks in the raw digits, with the following exceptions. Hits have a maximum length, and waveforms that dip below a fraction of their maximum and go back up again will be divided into two or more hits. The hit finder is implemented in `Reco/CompressedHitFinder_module.cc`, and configuration parameters are available in `Reco/CompressedHitFinder_module.cc`. Tuning of the maximum hit length and hit-splitting parameters has not yet been done, and further ideas to improve the hit finding are welcome. Once noise is simulated, a close look at the hit finding will be required. The hit finder produces data of the form `gar::rec::Hit` which is defined in `ReconstructionDataProducts/Hit.h`.

A second step of hit processing is clustering of hits to reduce their number and improve their spatial resolution. Nearby hits in space and time are grouped, and their charge centroids found. The module performing this task is tt Reco/TPCHitCluster_module.cc, and its configuration parameters are in `Reco/TPCHitCluster.fcl`. The output data product is `gar::rec::TPCCluster` which is defined in `ReconstructionDataProducts/TPCCluster.h`, and also associations between TPC clusters and hits. Most algorithms that process data from the tracker ought to use `gar::rec::TPCClusters` instead of hits. The BackTracker on the other hand, assumes that hits are localized to channels, while `TPCClusters` are not.

VII. TRACK PARAMETERS

The following discussion on track pattern recognition and fitting requires a definition of the track parameters. In a typical low-mass magnetic spectrometer, tracks are represented, at least locally, as helices. Five parameters are needed in order to specify a helix, and a sixth parameter is needed in order to locate a specific point on the helix. In a very high density tracking environment, however, scattering and energy loss are important, and a single set of helix parameters is inadequate to describe a track’s trajectory.

`GArSoft` uses a right-handed coordinate system, with y pointing upwards, x pointing along the E and B fields, and z pointing 101 mrad upwards of the beam axis, in the downstream direction. Since E points both along the positive and the negative x directions, the choice of x pointing predominantly southwards provides a right-handed coordinate system. The units of spatial coordinates are cm, and the units of time are microseconds.

We are usually most interested in a track’s parameters at the primary vertex, in order to establish the momentum magnitude and direction at the production point. At the track-finding stage, however, vertices have yet to be found. In a typical collider experiment, it is often assumed that vertices inside the beam pipe are the most interesting and thus track directions are chosen assuming that they travel from the inside of the detector to the outside. In the DUNE HPgTPC ND, primary vertices are distributed throughout the volume, and so either end of a track may be its origin. One may use energy loss and change of curvature in order to choose a beginning and end of a track, but this technique is imperfect due to scattering and measurement errors. Therefore, tracks have two sets of track parameters reported, one at one end, and one at the other. We are often interested in extrapolating tracks to the ECAL, and so both sets of track parameters are useful for downstream analysis.

The five track parameters are $(y_0, z_0, c, \phi_0, \lambda)$. In the software, the subscripts are often omitted, but are given here for clarity. Figure 1 shows the parameters y_0, z_0, $c = 1/r$, and ϕ_0, the parameters in the (z, y) plane. The curvature c is $1/r$ and its sign is significant. Because the direction of the track is unknown, however, the sign of the curvature does not determine the charge of a particle. The sixth track parameter is chosen to be x_0 to define a specific point on the helix. It is often stored outside of vectors or arrays of track parameters as it is not included in error calculations, being an independent variable. The point (x_0, y_0, z_0) is on the helix. An arbitrary point on the helix is given by

$$x = x_0 + r \tan\lambda(\phi - \phi_0) \tag{1}$$

$$y = y_c - r\cos\phi \tag{2}$$

$$z = z_c + r\sin\phi \tag{3}$$

The location of the center of the circle (y_c, z_c) is given by

$$y_c = y_0 + r\cos\phi_0 \tag{4}$$
$$z_c = z_0 - r\sin\phi_0 \tag{5}$$

The slope s of a track is defined to be

$$s = \frac{\sqrt{dy^2 + dz^2}}{dx} = \cot\lambda \tag{6}$$

Early versions of `GarSoft` used s instead of λ as the fifth track parameters, but this was problematic as s diverges for particles traveling perpendicular to the E and B fields (and along the neutrino beam direction!), and changed sign, creating a discontinuity in the mapping between track parameters and track angles where a lot of physically important tracks are. The λ parameter provides a continous, bounded mapping for these tracks and does not diverge for tracks parallel to the fields.

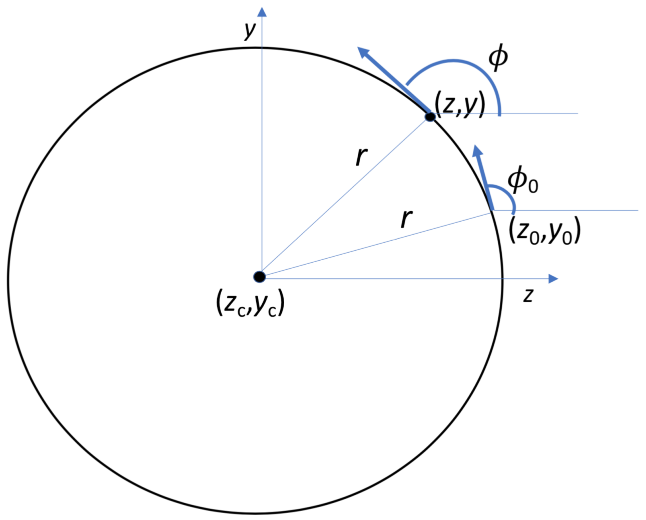

FIG. 1: Track parameter definitions in the (z, y) plane.

Track information is stored in the event record using the `gar::rec::Track` data product. It contains the six track parameters at both ends of the track as well as 5×5 error matrices, the track length and chisquared. Tracks have associations with `gar::rec::TPCCluster`s in the event, which reduces duplication of data in the event.

VIII. PATTERN RECOGNITION

Identifying tracks in `GArSoft` proceeds in two steps. First, track segments, called vector hits, are found, using `Reco/tpcvechitfinder2_module.cc`, which is configured by `Reco/tpcvechitfinder2.fcl`. The inputs are `gar::rec::TPCCluster` data and the output is `gar::rec::VecHit` data products as defined in `ReconstructionDataProducts/VecHit.h`. Associations between vector hits and TPC clusters are also produced. The second step is to cluster vector hits together into track candidates.

Only TPC clusters within a radius $R = 243$ cm are considered for vector hit formation due to the distortions on the ends resulting from a cylindrical drift volume and an 18-gon sensitive detector chamber. Charge drifting outside of the last pad row is registered on the last pad row. A real field cage would also be an 18-gon however and this restriction on radius may be relaxed once the simulation has the right shape for the field cage.

Vector hits are identified using linear fits to TPCClusters as functions of (x, y, z). The goal is to optimize the homogeneity and isotropy of the efficiency for identifying vector hits. Groups of hits are clustered together if they lie on a line segment no more than 10 cm long. A hit is clustered to a vector hit if it has a distance less than 2 cm from the line segment defining the vector hit. Once a hit is added, the line segment is re-fit. Six linear 2D fits are performed, two for each pair of coordinates, choosing either one as the independent variable. In order to define the

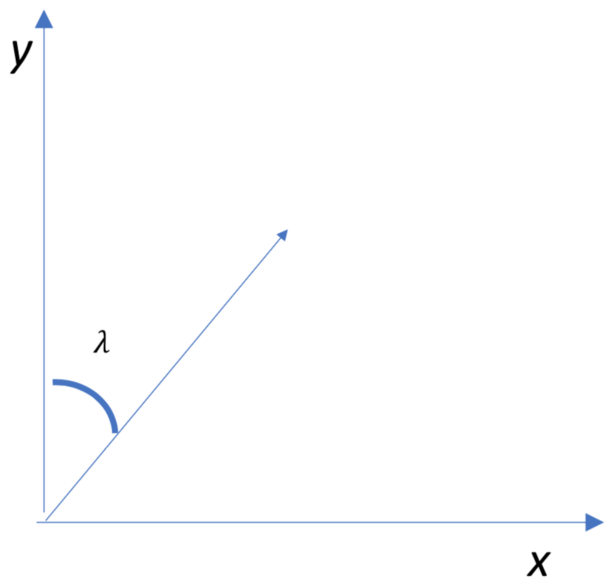

FIG. 2: Track parameter definitions in the (x, y) plane.

vector hit direction in 3D, the independent coordinate of the fit is chosen to be the one with the smallest sum of absolute values of slopes.

Two passes are performed for vector-hit finding. Vector hits with chisquareds exceeding a fixed threshold have their hits returned to the unassigned group, and the hit-assignment procedure is repeated, allowing these hits either to be assigned to other, existing vector hits, or to form new vector hits.

Once vector hits are found, they are clustered to form track candidates. The module that does this is `Reco/tpcpatrec2_module.cc`, and it is configured with `tpcpatrec2.fcl`. It taks the vector hits and their associations with TPC Clusters as inputs, and produces `gar::rec::Track` data products, as well as associations between the tracks and vector hits and TPC clusters. A vector hit is added to a vector-hit cluster if it matches all of the following criteria for one other vector hit already in the cluster. The notation used for the candidate vector hit is that its position is $\vec{p}_c$ and its direction is $\vec{v}_c$, while the test vector hit in the cluster has position $\vec{p}_i$ and its direction is $\vec{v}_i$. The search for matching vector hits in the cluster loops over the index i of vector hits in the cluster. The absolute sign of the direction of the vector hit is not significant, and the matching criteria take this into account.

The dot product of the directions must match within 0.9:

$$|\vec{v}_c \cdot \vec{v}_i| < 0.9. \tag{7}$$

The vector hit centers must be within 60 cm of each other:

$$|\vec{p}_c - \vec{p}_i| < 60 \text{ cm}. \tag{8}$$

Each vector hit must point near the center of the other – the "miss distance" is required to be less than 6 cm.

$$|\,(\vec{p}_c - \vec{p}_i) \times \vec{v}_i| < 6 \text{ cm}, \tag{9}$$

and

$$|\,(\vec{p}_c - \vec{p}_i) \times \vec{v}_c| < 6 \text{ cm}. \tag{10}$$

The two-dimensional η test is a model of vector hits lying on a circle. The average of the directions must point along the displacement between the centers. Here $\hat{x}$ is a unit vector along the x direction, which points along the E and B fields, and $\hat{y}$ and $\hat{z}$ are unit vectors pointing along the y and z directions, respectively. Define

$$\vec{t} = \hat{y}\,(\vec{p}_i - \vec{p}_c) \cdot \hat{y} + \hat{z}\,(\vec{p}_i - \vec{p}_c) \cdot \hat{z} \tag{11}$$

as the two-dimensional displacement between the vector hit centers, and

$$\vec{u}_i = \frac{\hat{y}\vec{v}_i \cdot \hat{y} + \hat{z}\vec{v}_i \cdot \hat{z}}{|\hat{y}\vec{v}_i \cdot \hat{y} + \hat{z}\vec{v}_i \cdot \hat{z}|} \tag{12}$$

and

$$\vec{u}_c = \frac{\hat{y}\vec{v}_c \cdot \hat{y} + \hat{z}\vec{v}_c \cdot \hat{z}}{|\hat{y}\vec{v}_c \cdot \hat{y} + \hat{z}\vec{v}_c \cdot \hat{z}|} \tag{13}$$

as the normalized two-dimensional unit vectors pointing along the vector hits in the (y, z) plane. Define the signed direction sum to be

$$\vec{s} = \vec{u}_i + \frac{\vec{u}_c \cdot \vec{u}_i}{|\vec{u}_c \cdot \vec{u}_i|}\vec{u}_c \tag{14}$$

And the η variable is defined to be

$$\eta = \left| \frac{\vec{s}}{|\vec{s}|} \times \vec{t} \right|. \tag{15}$$

The condition on η is that it be less than 1.2 cm to match a pair of vector hits. If a vector hit matches a vector hit in a cluster, but another vector hit within 20 cm of the test vector hit fails the requirement on η, then the vector hit match is dropped.

The final criterion is a match of λ between vector hits. Here λ is computed as

$$\lambda_i = \mathrm{Tan}^{-1}\left(\frac{|\hat{x} \cdot \vec{v}_i|}{\sqrt{(\hat{y} \cdot \vec{v}_i)^2 + (\hat{z} \cdot \vec{v}_i)^2}} \right) \tag{16}$$

for the test vector hit in the cluster, and similarly for the candidate vector hit:

$$\lambda_c = \mathrm{Tan}^{-1}\left(\frac{|\hat{x} \cdot \vec{v}_c|}{\sqrt{(\hat{y} \cdot \vec{v}_c)^2 + (\hat{z} \cdot \vec{v}_c)^2}} \right). \tag{17}$$

The λ values must match within 0.05:

$$|\lambda_i - \lambda_c| < 0.05 \tag{18}$$

An additional requirement on the relative signs of the slopes is applied.

$$(\vec{v}_c \cdot \vec{v}_i)(\vec{v}_c \cdot \hat{x})(\vec{v}_i \cdot \hat{x}) > 0. \tag{19}$$

This last requirement is only applied if the absolute values of the x components of $\vec{v}_i$ and $\vec{v}_c$ both exceed 0.01, so as not to discard matches because of a loss of significance of the sign of the x components of the directions.

Some concerns about the performance of the pattern recognition stage include

- Incorrectly joining two tracks together: Most vertices will have multiple charged-particle tracks leaving them. Since the first step of pattern recognition has not yet identified the primary vertex candidates (tracks are needed to identify vertices), a track on one side of the primary vertex can be joined incorrectly with a track pointing in the opposite direction but with a small angle. Another category of mistakes is the joining of the electron track and the positron track in a photon conversion. At the conversion point, the two tracks coincide in position and direction, and so a vector hit which starts at the conversion point is consistent with both tracks and causes the two sets of vector hits to be clustered erroneously. This is an easier case to resolve than the primary vertex case as the two tracks bend in opposite directions and quickly become distinct.

- Splitting long tracks into smaller pieces: A track may have a scatter ("kink") partway along it, or a delta ray may be emitted, or some TPC clusters from another track may get misassigned to a vector hit, causing it to point in the wrong direction or to be displaced from the track. All of these cause vector-hit clusters to be broken into smaller pieces, presenting more candidate tracks to the downstream fit than there are true charged particles. This effet is currently quite noticeable near the primary vertex, where hits from more than one track can be confused and the vector hits near the primary may point in the wrong directions, breaking the tracks.

One way to improve the performance of the pattern recognition is to allow the fitter to reassign TPC clusters to tracks, or to make a second pass through hit assignment once the track parameters have been estimated.

The `gar::rec::Track` data product contains track parameters and uncertainties quoted at both ends of the track. These are computed using the standalone `C` routine `gar;:rec::initial_trackpar_estimate` which calculates the track parameters from three three-dimensional points along the track. These points are currently taken to be the track endpoint, the 50th TPC cluster if available, and the 100th TPC cluster, if available. If fewer than 100 TPC clusters have been assigned to the track, then the last TPC cluster and the TPC cluster midway from the first to the last TPC cluster are used.

In order to define which TPC clusters are the track endpoints and which ones are 100 TPC clusters away from the endpoints, a TPC cluster sorting algorithm is defined. The first step in finding the endpoints of a track is to identify the six TPC clusters that have the minimum and maximum x, y, and z coordinate values. Then the sum of the distances from each of the six candidate endpoint TPC clusters to each other TPC cluster is computed, and the TPC cluster for which this distance sum is largest is chosen as one of the track endpoints. The endpoint is the first element in the sorted TPC cluster list. Sorting proceeds by adding the hit that minimizes a linear combination of the distance along the current estimate of the track direction and perpendicular to it, with a coefficient of 0.1 on the perpendicular component, relative to the longitudinal component. The current direction is determined from 10 cm of hits. If a track segment in the sorting algorithm is not yet 10 cm long, the TPC cluster that is closest to the growing endpoint is chosen as the next TPC cluster to add.

The reverse sort is the inverse of the forwards sort, and is used to define the track parameter estimate on the other end of the track.

IX. TRACK FITTING

The fitter uses an Exended Kalman Filter [7] in order to compute the best estimates of the track parameters on both ends of the track. As the associations between tracks and TPC clusters does not preserve ordering, the TPC clusters are re-sorted using the same algorithm used at the end of the pattern recognition, and the initial track parameter estimates are recomputed. The reason for doing this twice is that the Kalman filter may be iteratively applied with different TPC cluster assignments, and each time the track parameters must be initialized.

Two fits are performed per track, a forwards fit and a backwards fit. Due to a historical convention in the labeling of members of `gar::rec::Track`, one of the track ends is called the "Vertex" end and the other is the "End" end. Because tracks must be found and fit before vertices are identified, and because it is not known a priori which end of a track has the vertex on it (if either one), these labels are arbitrary. Since the Kalman filter steps along the track, using a Bayesian update of the track parameters as it goes along, the track parameters are evaluated at the opposite end to the end the track starts on. The "forwards" fit therefore calculates the "End" track parameters, and the "backwards" fit calculates the "Vertex" or "Beginning" track parameters. The notation "forwards" and "backwards" is used to label the length and chisquared members of `gar::rec::Track`.

As a design issue for *art* data products, `gar::rec::Track`'s class defines as few methods as possible [8]. A useful class for holding track information however ought to supply many tools for calculating things like the distance from a point to a track, and extrapolating the track beyond its endpoints. In order to separate the calculation methods from the persisted object, a class called `gar::rec::TrackPar` is defined to hold the utility methods. Instances of this class are not stored in the *art* event, but methods for making a `gar::rec::Track` from a `gar::rec::TrackPar` and vice versa are provided.

The steerable parameters of the Kalman filter include the step uncertainties that are added to the track parameter error matrix before each new point is added. These parameterize how "stiff" the fit is. The TPC clusters scatter around the true track location due to diffusion and the discretization of the pads. A more complete field response is expected to smooth out the discretization and reduce the point scatter. Fitting a helix to the last three TPC clusters is very likely to give random track parameters, especially with large curvatures. The step uncertainty parameters de-weight hits farther away from the track endpoint at which the track parameters are to be evaluated, which is important due to scattering and energy loss. These parameters therefore require some optimization. The more hits which contribute to the parameter measurement reduce the fluctuations due to hit position spread, but increase the fluctuations due to scattering and energy loss. If the scattering is large, a track may be split, improving the resolution.

The momentum resolution for muons in GENIE events is given in Sec XIV.

The track fitter currently contains an option for dropping TPC clusters that are too far from the track, and it fills a vector of unused TPC clusters. The current setting of the width of the road however is very long, disabling this feature. A second pass through the tracking, allowing TPC clusters to be reassigned, would improve the issues identified above with the pattern recognition.

X. VERTEX FINDING AND FITTING

The vertex-finding module is `vertexfinder1_module.cc` and it is configured with `vertexfinder1.fcl`. This module takes as input `gar::rec::Track` data products and produces `gar::rec::Vertex` data products and associations between the tracks and vertices. A track endpoint is a candidate for vertexing with another track endpoint if they are within 12 cm of each other. Vertices are fit with a linear extrapolation from the endpoints of each track. A list of found vertices and tracks is kept so that a track is not vertexed with itself even if it loops around in the magnetic field, and two tracks cannot be vertexed with each other more than once. Tracks, and even track ends, may contribute to more than one candidate vertex.

There is still work to be done with the vertex finder: using helical extrapolations of tracks from the ends, calculating the covariance matrix of the vertex spatial coordinates, and improving the track association selection criteria are all things that can improve the vertexing.

XI. BACKTRACKER

The BackTracker service is located in the `MCCheater` directory in `GArSoft`. It is largely copied from the corresponding LArSoft BackTracker. The purpose of the BackTracker is to assign MC truth particles to reconstructed hits. It has significant use in wire-based liquid argon TPC reconstruction where tracks and showers overlap and the data are highly ambiguous. In a 3D pixel gaseous argon TPC however, comparing reconstructed quantities with the true MC quantities is easier as the tracks have a high degree of purity of correct hits, and missing hits and misassigned hits cause local distortions in the tracks. Nonetheless, the service is present, and users are encouraged to try it out. The BackTracker assumes that hits are on one DAQ channel, and so it does not work with TPC clusters, which average over several hits.

There is a `gar::rec::Shower` data product but it is not used by the TPC tracking. The ECAL has separate data products for storing CaloRawDigits, CaloHits and Clusters.

XII. EVENT DISPLAYS

There are two event displays provided with `GArSoft`. One is based on `NuTools`'s `View3D`, and is invoked with `art -c evd.fcl artrootfile.root`. By default, it draws a wire frame outline of the TPC, MC particles, and Tracks with associated TPC clusters in different colors. Drop-down menu items can configure the drawing to show hits, TPC clusters, and vector hits. A separate raw mode shows raw digits as 1-cm-long green lines. A tracking dE/dx display (called "calorimetry" after the LArSoft version) exists but has not been developed – it shows liquid argon dE/dx vs residual range curves and no data in a 2D pad. It may be of use when selecting a track and comparing its dE/dx values against those of different particle hypotheses. The `View3D` display currently does not show any information from the ECAL.

A second event display, based on `ROOT`'s `TEve` features, is available. It is invoked with `art -c evd3D.fcl artrootfile.root`. It shows MC particles, energy deposits, and ECAL information. `TEve` might not be compatible with all graphics hardware or video drivers. A way to test if `TEve` works with current video drivers on your computer is to exercise the examples in `$ROOTSYS/tutorials/eve`. These tutorials fail on an (old) SLF7 desktop with the display on the desktop's X server, but succeed over the network when the display is sent back to the same desktop, presumably because rasterization is being done in software in the latter case.

XIII. ANALYSIS NTUPLE

In order to make ntuple analysis easier, a module `anatree_module.cc` is provided. It is configured with `anatree.fcl`, and run with `anajob.fcl` in a separate job that takes reconstructed rootfiles as input. The `anatree` contains information from `MCTruth`, `MCParticle`, `gar::rec::Hit`, `gar::rec::Track`, and `gar::rec::Vertex` data products and associations, and places them in a flat `ROOT` tree that can be analyzed without `GArSoft` tools. Switches in the `fcl` parameters control whether to keep or drop various branches.

A second analysis tree, is made by `CaloAnaTree_module.cc`, which is configured with `CaloAnaTree.fcl` and run with `caloanajob.fcl`. The `CaloAnaTree` contains information from `MCTruth`, `MCParticle`, `CaloDeposit`, `CaloRawDigit`, `CaloHit`, and `Cluster` data products.

A future direction is to make an analysis tree that contains both tracking and calorimetry information so that analyses needing such quantities can proceed.

XIV. PERFORMANCE

The performance of the TPC tracking on GENIE events is summarized below. A sample of 2 GeV/c ν_μ and ν_e's generated along a line along the z direction 10 cm from the cathode is used to test the tracking and vertexing performance. The tracking efficiency and resolution parameters are given as functions of total momentum and the angles in 3D. The muons in ν_μCC events are of particular interest, partly because of the importance of reconstructing them well, and partly because muons interact rarely in the detector.

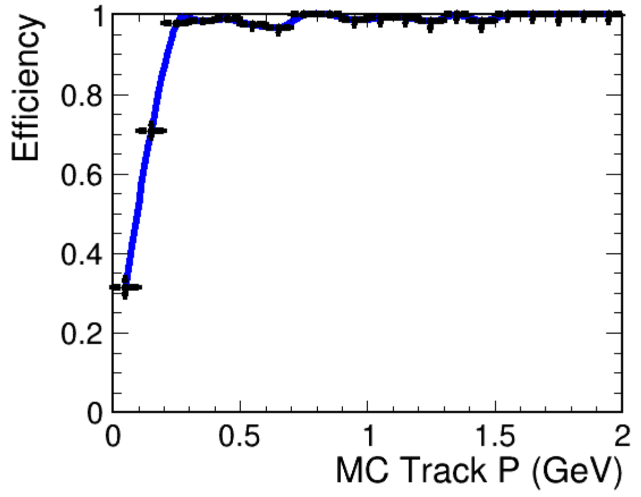

FIG. 3: Tracking reconstruction efficiency as a function of total momentum p.

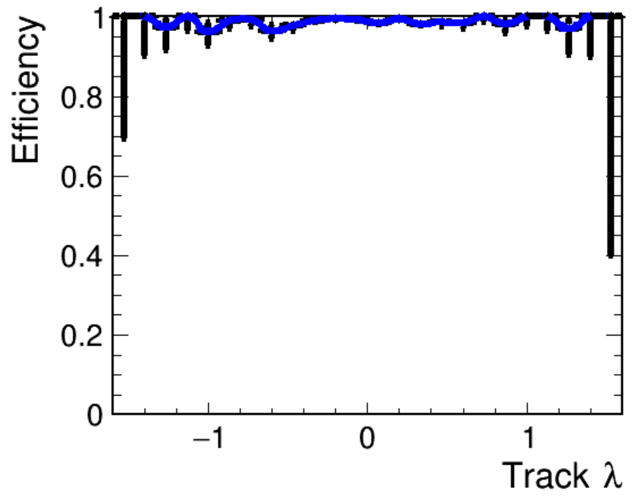

FIG. 4: Tracking reconstruction efficiency as a function of λ for tracks with total momentum $p > 200$ MeV/c.

XV. SUMMARY

The current status of tracking in `GArSoft` is summarized in this note, covering simulation and reconstruction of TPC data. The ECAL components of the software are only briefly touched on and will be described in a separate document.

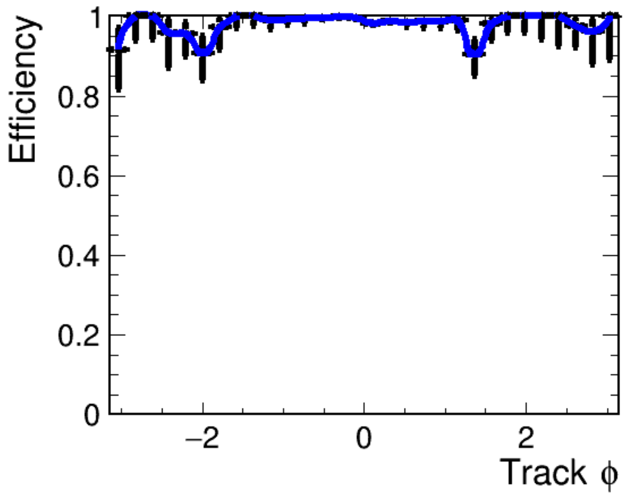

FIG. 5: Tracking reconstruction efficiency as a function of ϕ for tracks with total momentum $p > 200$ MeV/c.

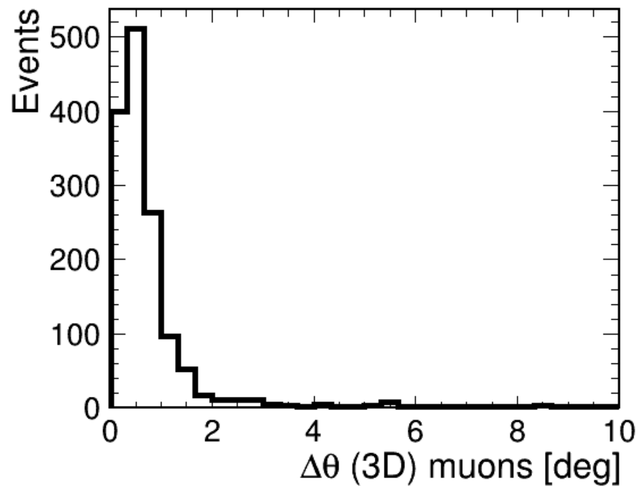

FIG. 6: Muon angular resolution at the creation point for ν_μCC events.

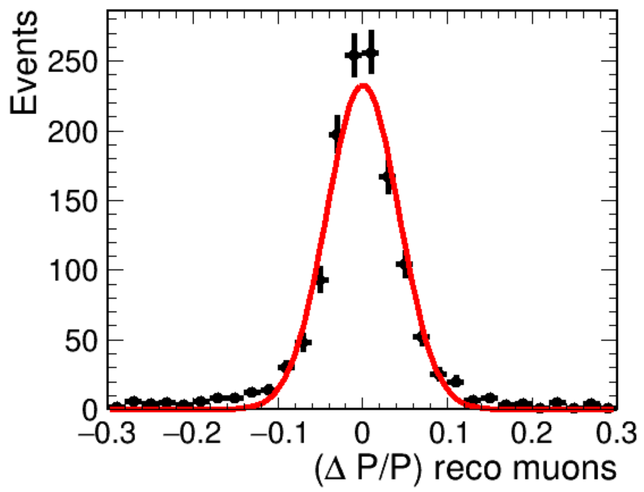

FIG. 7: Muon momentum resolution at the creation point for ν_μCC events.

[1] The LArSoft toolkit: https://larsoft.org/
[2] The *art* framework: https://cdcvs.fnal.gov/redmine/projects/art
[3] https://cdcvs.fnal.gov/redmine/projects/garsoft
[4] https://cdcvs.fnal.gov/redmine/projects/mrb/wiki
[5] https://cdcvs.fnal.gov/redmine/projects/ups/wiki/Documentation
[6] https://cernvm.cern.ch/portal/filesystem
[7] https://en.wikipedia.org/wiki/Extended_Kalman_filter
[8] https://cdcvs.fnal.gov/redmine/projects/art/wiki/Data_Product_Design_Guide