Water Cherenkov Detector Reconstruction Tools

Andy Blake, Cambridge University.

Reconstruction Workshop,
December 2011.

Overview

- **This talk provides an overview of my framework for reading in WCSim ROOT files and running reconstruction routines.**
- **It's just an interim framework, but provides the following:**
  - Reads in the WCSim events, geometry and truth information, and provides lookup methods for all of them.
  - Contains some simple data classes for reconstructed quantities.
    - Not a complete set of data classes or variables, but a start.
  - Has a simple framework for running reconstruction algorithms, and also some simple algorithms.
    - Not flexible, needs development. However, code works!
  - Also contains framework for event displays and output ntuples.
- **Code in SVN, and provides base for Common Reconstruction.**

https://lbne.bnl.gov/svn/people/CommonReconstruction/WCSimAnalysis

With apologies for unimaginative name, and lack of documentation

(1) Building the Code

Instructions

- ◇ **Contains code from: AB, Matt Wetstein, Gavin Davies, Tian Xin (and others – apologies to those missed out of this list!)**
- ◇ **Dependencies: WCSim, ROOT (WCSim also depends on Geant4).**

Instructions:

(1) Check out code:

> svn co https://lbne.bnl.gov/svn/people/CommonReconstruction/WCSimAnalysis/

(2) Set up environment.

- Same as WCSim, but one additional environment variable needed: WCSIM variable should point to top directory of WCSim package.
- WCSimAnalysis directory has example setupWCSimAnalysis.sh

(3) Build Code

> make (with apologies for the inevitable compiler errors…!)

(4) Try Running…

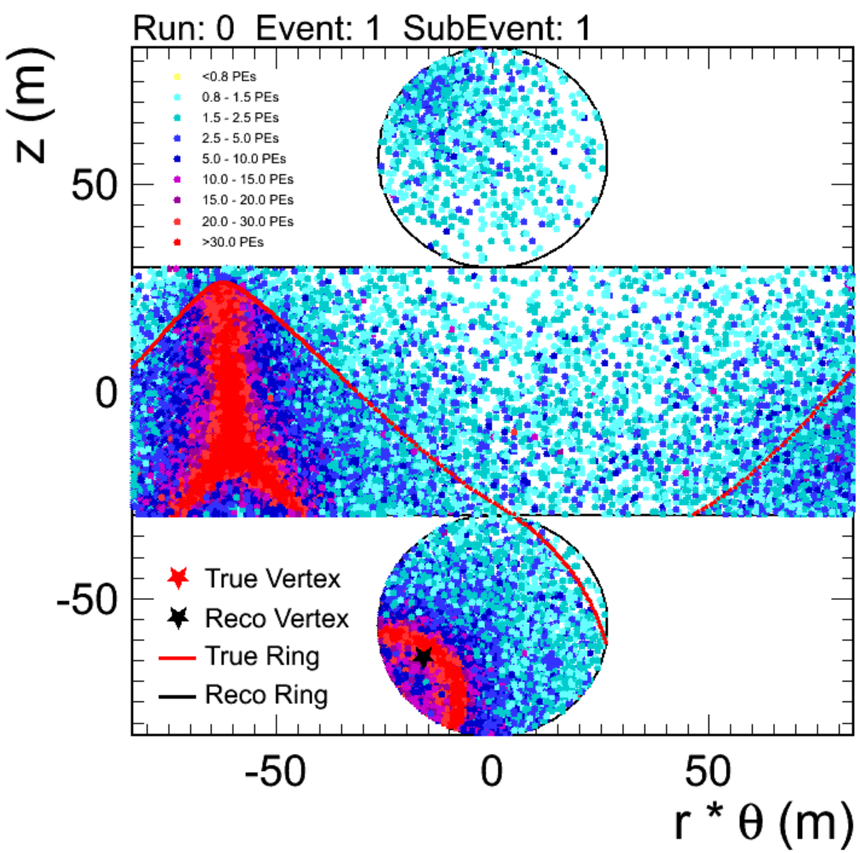
> root –l macros/wc_eventdisplay_3D.C

(5) Some examples of working code.

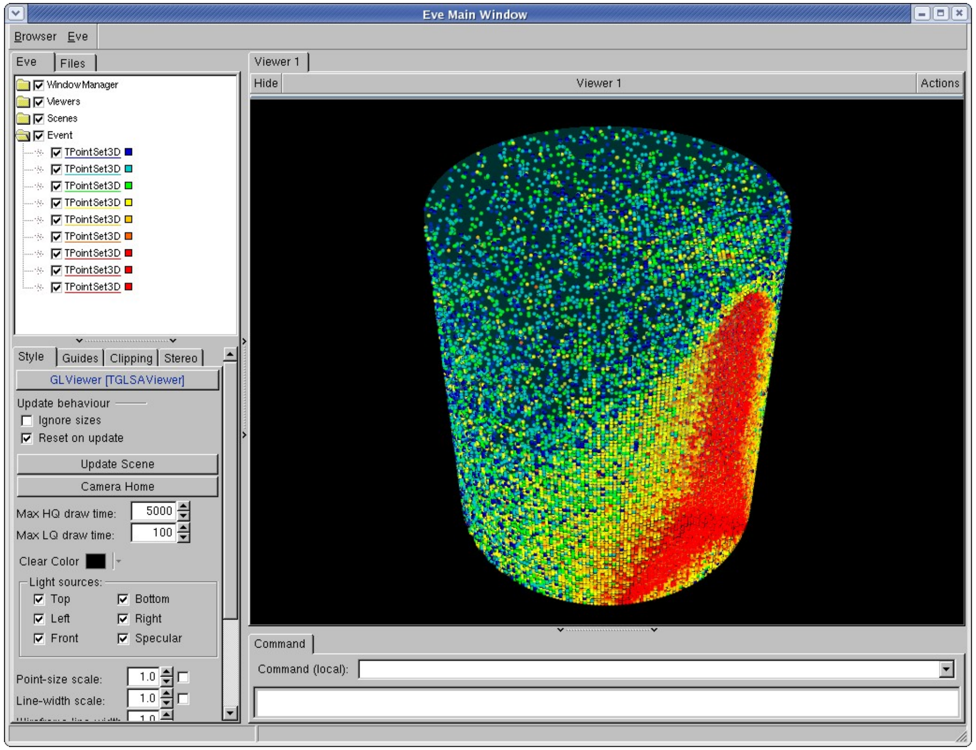
/lbne/app/users/trj/wcsimsoft/ OR /lbne/app/users/blake/software2010/

Event Displays

<u>Example</u>: through-going cosmic muon in 100 kton detector.

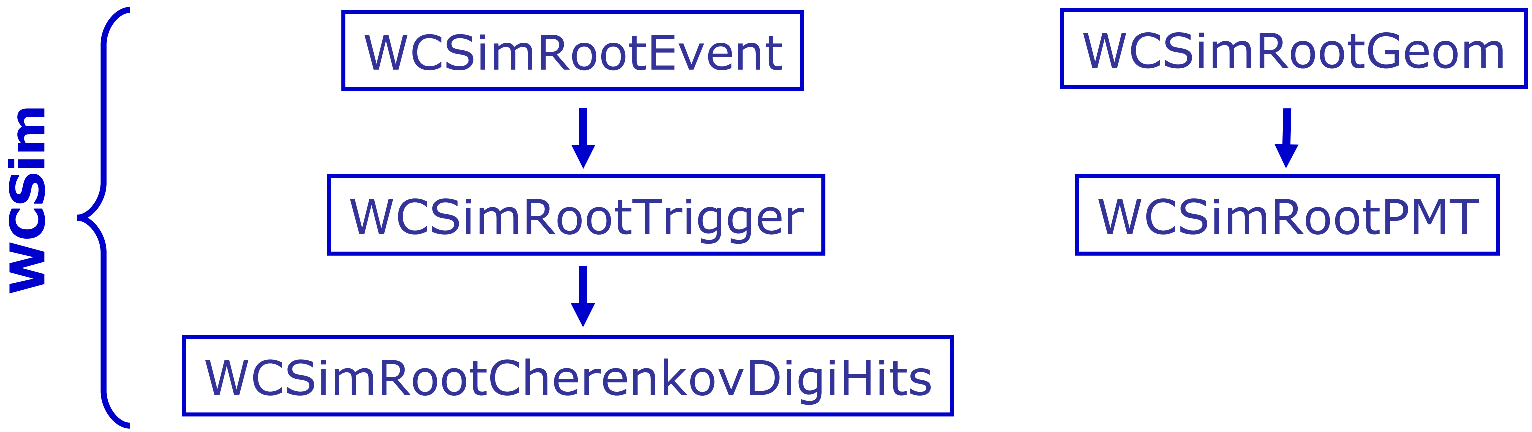


ROOT Display using TEve Classes



(2) Reconstruction Framework

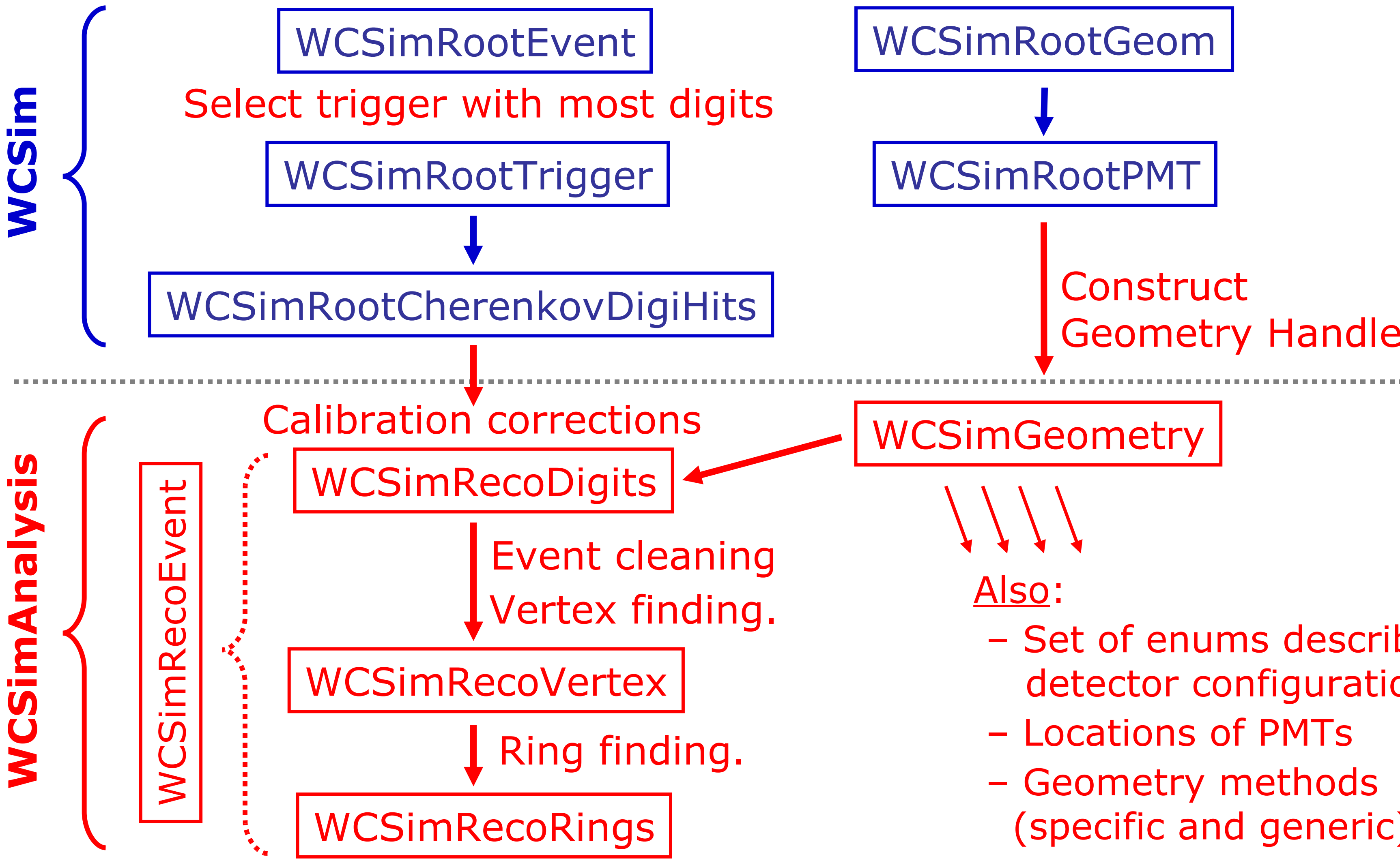
Framework for Reconstruction



WCSimAnalysis:

- Packages information for use by reconstruction.
- Provides a set of reconstruction data classes.
- Provides framework for running algorithms.

Framework for Reconstruction



Also:

- Set of enums describing detector configuration
- Locations of PMTs
- Geometry methods (specific and generic)

Framework for Reconstruction

WCSim

WCSimRootEvent

Select trigger with most digits

WCSimRootTrigger → WCSimRootCherenkovDigiHits

MC truth information → WCSimRootTracks

WCSimAnalysis

WCSimTrueEvent

- Select ‘visible’ tracks:
  - charged particles above Cherenkov threshold; and also photons.
  - Require:
    - ◇ dS > 25 cm
    - ◇ KE > 25 MeV.

WCSimTrueTracks

- Select ‘primary’ track:
  - above-threshold track with largest dS*KE (?)
  - store four-momentum and Geant4 vertex.
    - ◇ Also, estimate actual entry/exit positions.

Reconstruction Classes

WCSimRecoEvent

```
Int_t fRunNum;
Int_t fEventNum;
Int_t fTriggerNum;
```
Header Information

```
std::vector<WCSimRecoDigit*> fDigitList;
std::vector<WCSimRecoRing*> fRingList;

WCSimRecoVertex fVertex;
```

WCSimRecoRing

```
Double_t fVtxX;
Double_t fVtxY;
Double_t fVtxZ;
```
Vertex

```
Double_t fDirX;
Double_t fDirY;
Double_t fDirZ;
```
Direction

```
Double_t fAngle;
Double_t fHeight;
```
Hough Parameters

WCSimRecoVertex

```
Double_t fX;
Double_t fY;
Double_t fZ;
Double_t fTime;
```
Vertex Position and Time

```
Double_t fDirX;
Double_t fDirY;
Double_t fDirZ;
```
Vertex Direction

```
Double_t fConeAngle;
```
Fit Parameters

```
Double_t fFOM;
Int_t fStatus;
```
Goodness of Fit

WCSimRecoDigit

```
Int_t fRegion;        Double_t fRawTime;
Double_t fX;          Double_t fRawQPEs;
Double_t fY;          Double_t fCalTime;
Double_t fZ;          Double_t fCalQPEs;
```

```
Bool_t fIsFiltered;
```
A ‘Good’ Digit

Reconstruction Classes

WCSimTrueEvent

```
Int_t fIpdg;
```
PDG code of neutrino, or primary track

<u>Also needs</u>:
Event Type…
Interaction Type…
etc.

```
// List of Above-Threshold Tracks
std::vector<WCSimTrueTrack*> fTracks;
```

WCSimTrueLight

Might want to store information from WCSim's list of photons

WCSimTrueTrack

```
Int_t fIpdg;
```
PDG code

```
Double_t fTrkP;
Double_t fTrkE;
// etc...
```
True Kinematics

```
// Geant4 Vertex and End Positions
Double_t fG4VtxX;   Double_t fG4EndX;
Double_t fG4VtxY;   Double_t fG4EndY;
Double_t fG4VtxZ;   Double_t fG4EndZ;

// Entry and Exit Positions in Detector
Double_t fVtxX;     Double_t fEndX;
Double_t fVtxY;     Double_t fEndY;
Double_t fVtxZ;     Double_t fEndZ;

Double_t fDirX;
Double_t fDirY;
Double_t fDirZ;
```
True Direction

```
Double_t fLength;
```
True Length.
Other?

Running Reconstruction

WCSim ROOT File (‘wcsim.root’)

Contains: Event Tree: ‘wcsimT’
Geometry Tree: ‘wcsimGeoT’

Running WCSimAnalysis:

```
// Load Data
WCSimInterface::LoadData(“wcsim.root”);

// Initialise Reconstruction
WCSimReco* myReco = WCSimRecoFactory::Instance()->MakeReco();

// Loop over Events
for( Int_t i=0; i<WCSimInterface::GetNumEvents(); i++ ){
    WCSimInterface::LoadEvent(i);   // Get i’th event

    // Build True and Reco Events
    WCSimTrueEvent* myTrueEvent = WCSimInterace::TrueEvent();
    WCSimRecoEvent* myRecoEvent = WCSimInterface::RecoEvent();

    // Run Reconstruction
    myReco->Run(recoEvent);
}
```

Running Reconstruction

WCSim ROOT File ('wcsim.root')

Contains: Event Tree: 'wcsimT'
Geometry Tree: 'wcsimGeoT'

Running WCSimAnalysis:

```
// Load Data
WCSimInterface::LoadData("wcsim.root");

// Create Ntuple Writer
WCSimNtupleWriter* myWriter = new WCSimNtupleWriter();

// Choose Type(s) of Ntuple
myWriter->UseNtuple("Reco");

// Run over Events
myWriter->Run(100); // run  over 100 events
```

(3) Reconstruction Algorithms

Reconstruction Tools

(1) <u>Data 'Cleaner'</u>

- Remove <1 PE digits. Then use clustering algorithm to remove outliers in space and time. This tags a set of 'good' digits.
- Apply overall cut on number of 'good' digits in event.

(2) <u>Vertex Finder</u>

- Reconstruct event vertex and direction using timing information, pulse heights of hits, and angular distribution of hits.
- Build up vertex in several stages.
  - ◇ First, find seed vertex using four-hit combinations.
  - ◇ Next, fit vertex and direction assuming point source.
  - ◇ Finally, run the fit assuming an extended source.
- Full vertex fit is 9-parameter minimisation, using Minuit (ugh!!!).
  - ◇ Position/Time (4), Direction (2), Shape parameters (3).

(3) <u>Ring Finder</u>

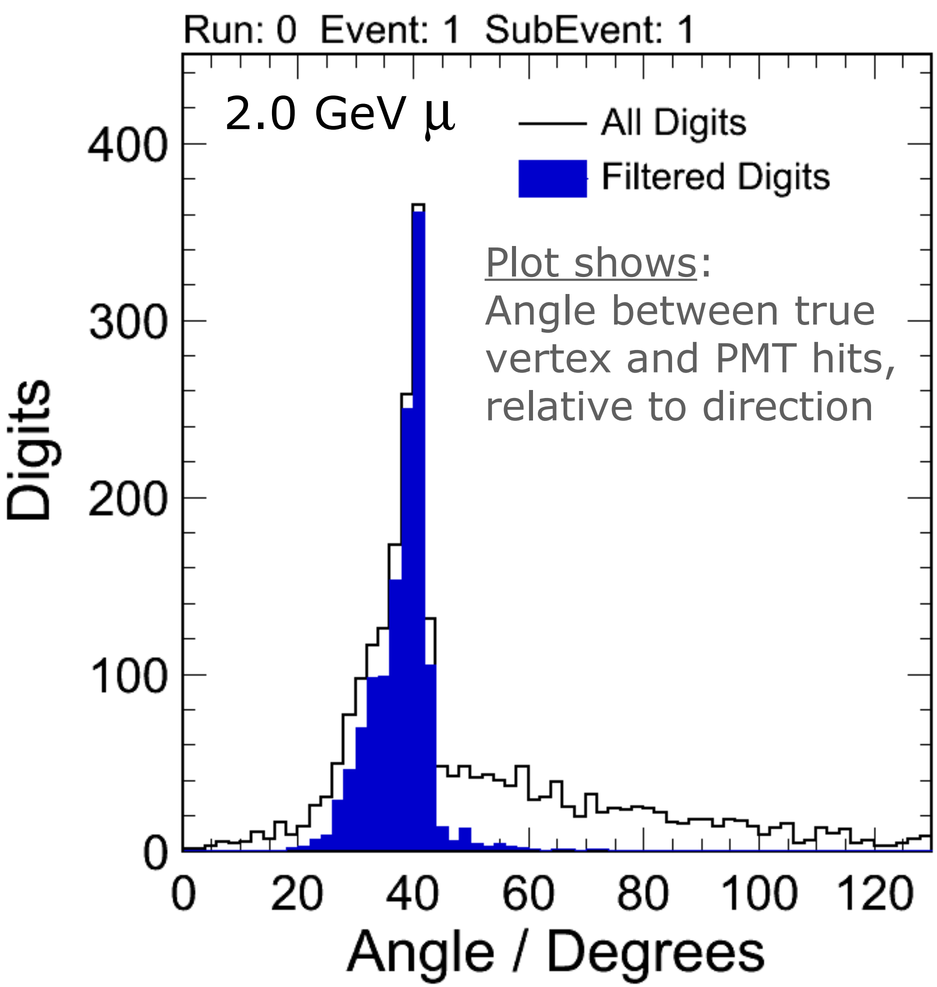
- Reconstruct single ring using 3-parameter Hough Transform.
  - ◇ Direction angles (2), Cone angle (1).

Tuning the Reconstruction

- **Each reconstruction algorithm must first be tuned using a training sample of events.**
  - Need a set of input parameterisations. These are constructed by hand, which is quite labour-intensive!
  - In general, parameterisations depend on detector geometry, event energy etc…
    - ◇ However, the code currently only supports a single set of parameterisations.
- **For the results shown here, reconstruction has been tuned using single-particle electrons and muons, for the case of 200 kton detector geometry.**
  - Samples: 1.2, 1.6, 2.0 GeV (both positives and negatives).
  - Detector: 12-inch PMTs, 10% coverage, no light collectors.
- **Would probably need to re-tune code for other samples or geometries. Would also help to bin by energy too.**
  - Need to automate the whole process somehow!!!

Data 'Cleaning'

Example Event:



- **Cherenkov rings embedded in background of late light, hampering reconstruction.**
- **Try to clean up events by picking out regions with highest density of hits.**
- **Applies following cuts:**
  - Reject all <1 PE digits.
  - Select the 'good' digits.
    - ◇ These digits must have >1 neighbouring digits within 2 m and 25 ns.
    - ◇ Note: not optimised!!
  - Require >50 'good' digits.
- **Pass the 'good' digits to the vertex fitter.**

Vertex Fitting Algorithm

POINT SOURCE

Seed Position ← Use four-hit combinations.

↓

Fitted Position ← Fit to timing information .

↓

Seed Direction ← Charge-weighted direction.

↓

Fitted Direction ← Fit to angular distribution.

↓

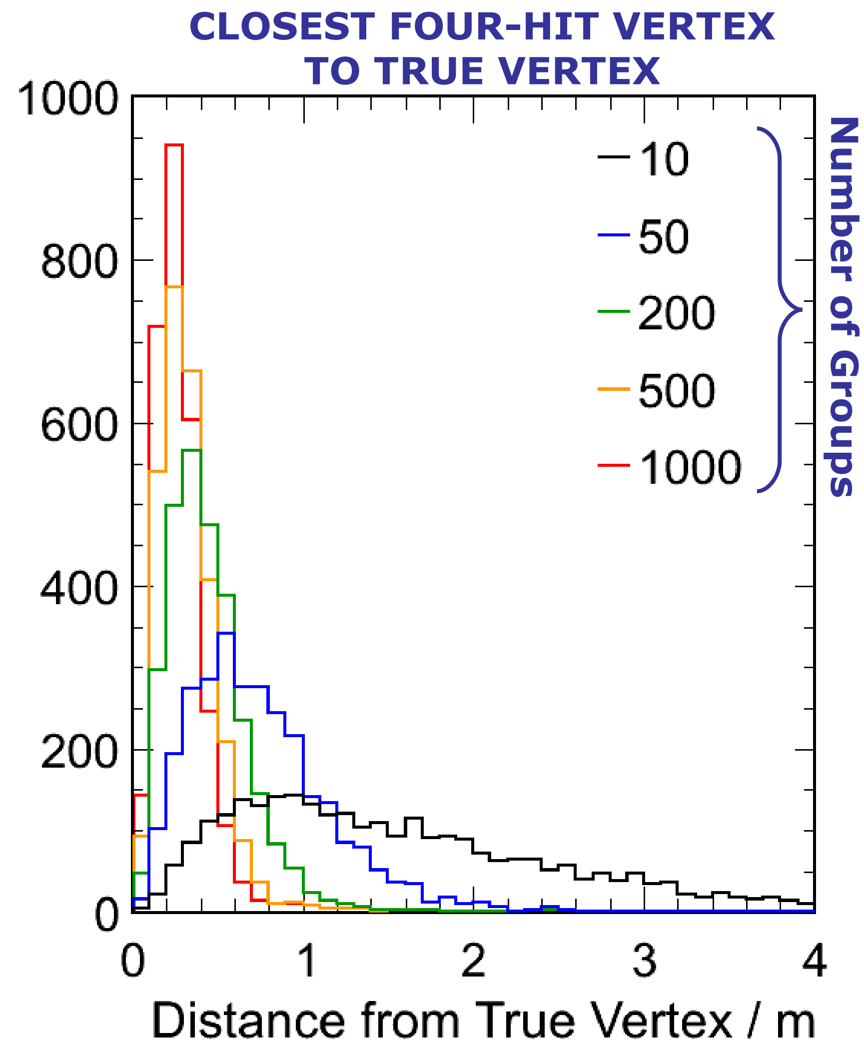
Full Vertex Fit (Point Source) ← Combined fit to timing and angular distribution

↓

Full Vertex Fit (Extended Source) ← Combined fit, incorporating Nuisance parameters.

Four-Hit Combinations

- **Use four-hit combinations to seed a vertex position.**
  - Four-hit combinations have common vertex position/time.
- **<u>Problem</u>: Since most GeV events have >1000 digits, there are an overwhelming number of combinations!!!**
  - To reduce combinations, only use hits with >5 PEs.
  - After that, select four-hit groups pseudo-randomly.
  - Using 200 four-hit groups, one is almost always within 1m of true vertex.
  - Pick vertex position which maximum likelihood.



Likelihood Function

- **Likelihood function divided into time and spatial parts:**

$$L = \prod_i P_t(t_i, q_i)\, P_q(\theta_i, q_i)$$

  - <u>where</u>: t_i, q_i are measured time and pulse height of i^{th} hit,
    θ_i is angle from vertex to i^{th} hit relative to direction.
  - The pdfs $P_t(t)$ and $P_q(\theta)$ are constructed by parameterising distributions of digits, based on true vertex and direction.
    - ◇ In other words, they represent the 'average' event.
  - However, pdfs can be adjusted on an event-by-event basis using a set of nuisance parameters.
- **<u>Note</u>: this isn't a full likelihood fit…**
  - Doesn't calculate a full hit-by-hit Monte Carlo expectation.
  - Parameterisations are constructed from final digitised hits.
    - ◇ Should really parameterise each component of simulation.
  - Use single set of functions, with no breakdown by energy etc…

Timing Residuals

- **The timing part of the likelihood function, $P_t(t)$, is actually a pdf in the timing residuals.**
- **For point source, timing residual of each digit is given by:**

$$\delta_i = \Delta t_i - \frac{\Delta r_i}{(c/n)}$$

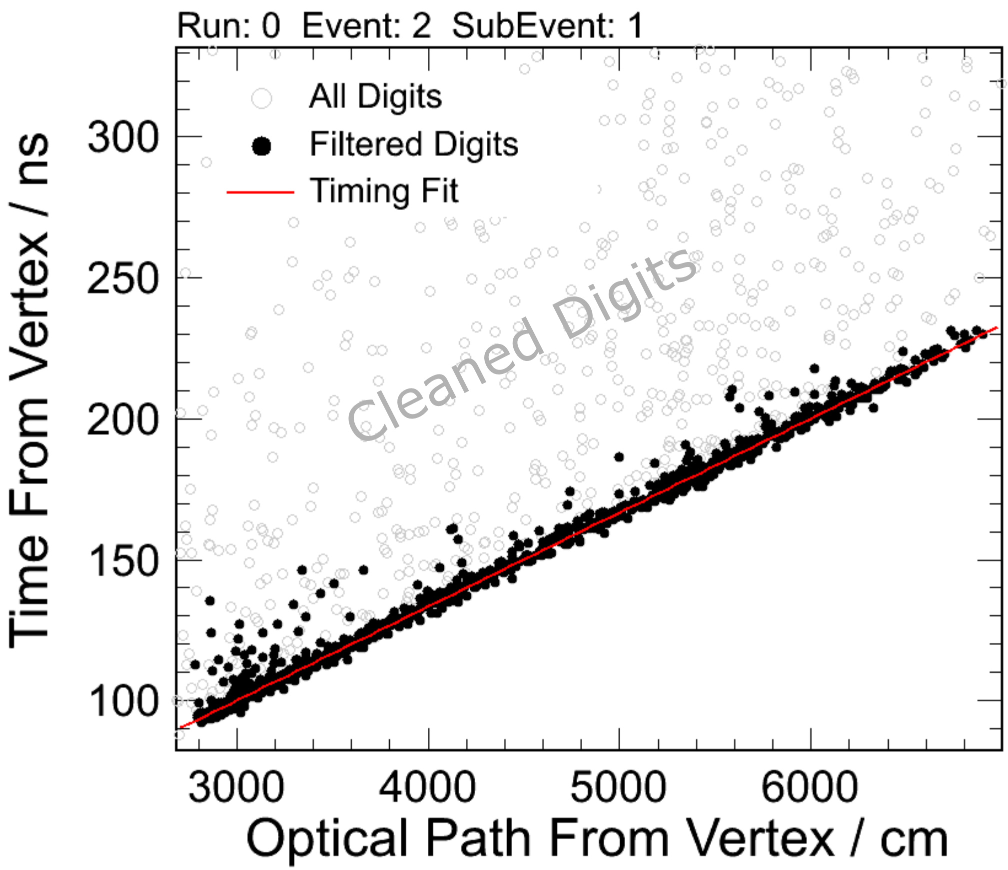
where: Δt = displacement in time from vertex to PMT.
Δr = distance from vertex to PMT.

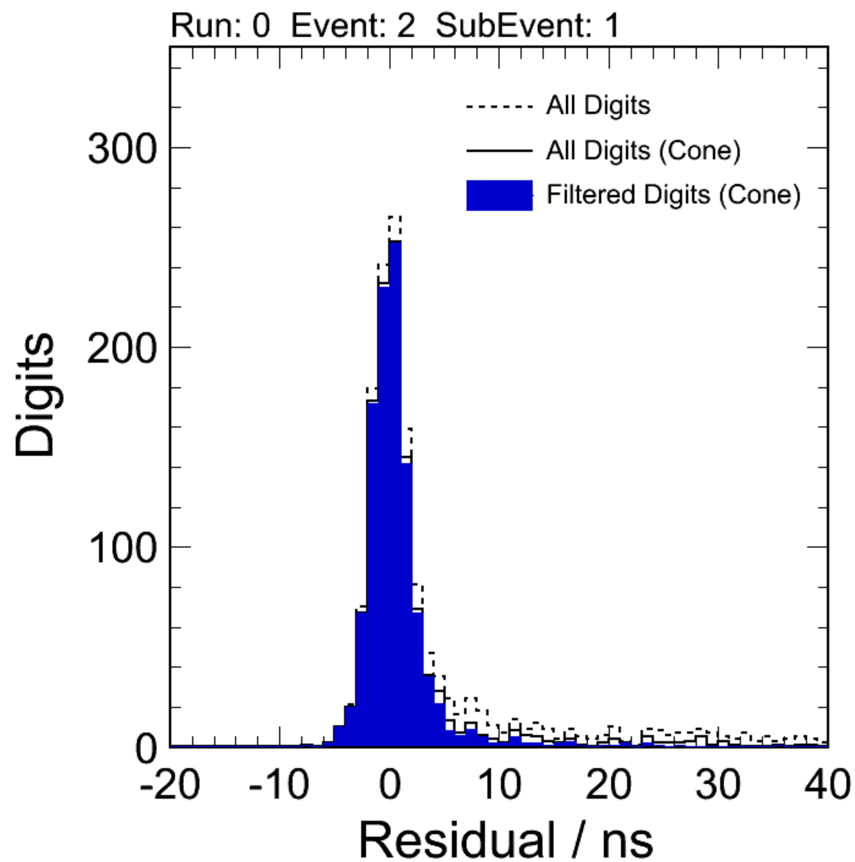
- **For extended source, timing residual of each digit given by:**

$$\delta_i = \Delta t_i - \frac{\Delta L_i}{c} - \frac{\Delta r_i}{(c/n)}$$

where: ΔL = distance along track to point of photon emission.
Δr = photon propagation distance to PMT.

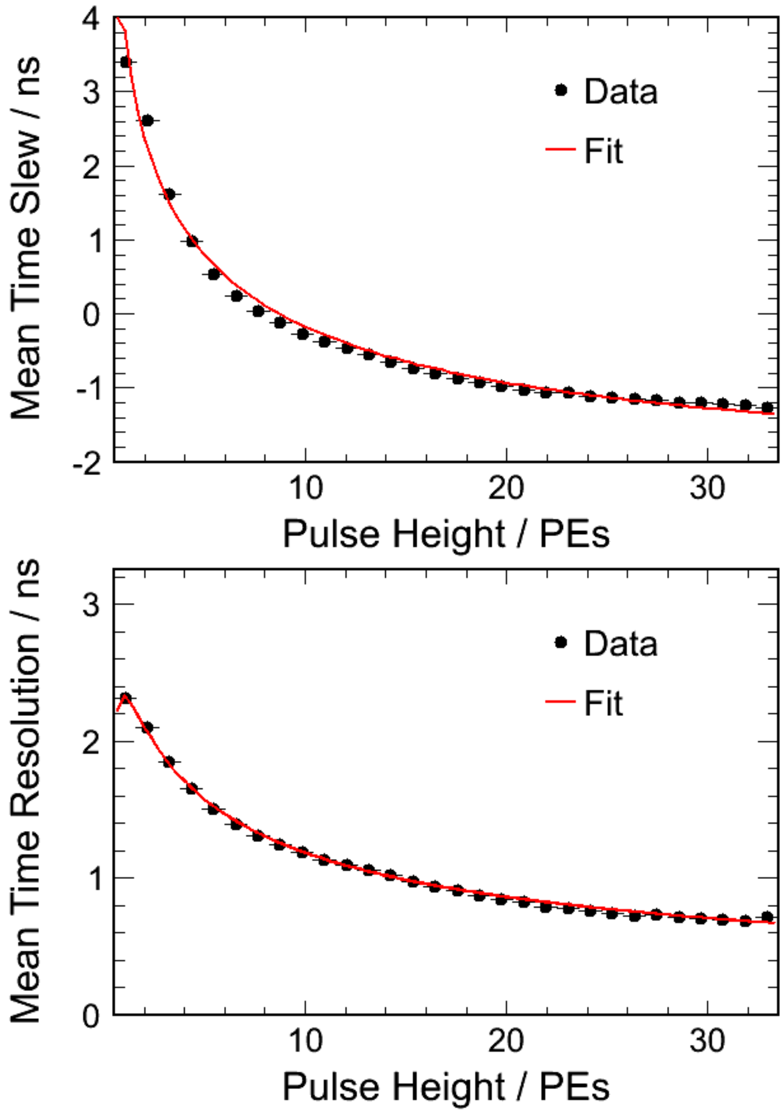
Timing Residuals





Fit vertex position by maximizing sharpness of timing residuals.

Timing Distributions



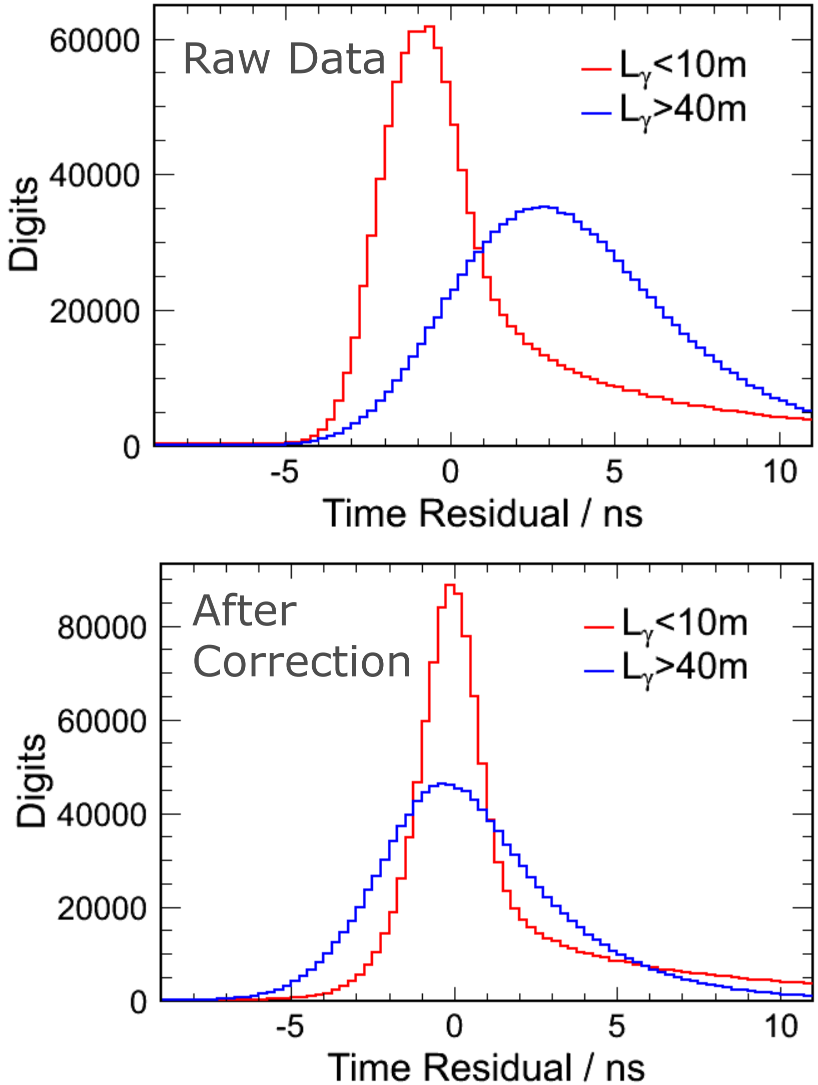
- **Obtain significant improvement in vertex resolution by correcting for timewalk, and by accounting for timing resolution in the fit.**
- **Timewalk and timing resolution are found by calculating time residuals of those hits inside Cherenkov cone, assuming the true vertex and direction.**
- **Timewalk given by mean residual. Parameterise as:**

$$\mu_T = c_0 + c_1 \log Q + c_2 (\log Q)^2$$

- **Resolution given by rms residual. Parameterise as:**

$$\sigma_T = c_0 + \frac{c_1}{\sqrt{Q}} + \frac{c_2}{Q}$$

Timing Distributions



- **Timing residuals also appear to become more delayed for longer photon propagation distances.**
  - Left plot shows time residuals for short (<10m) and long (>40m) photon propagation distances.
  - Later residuals for longer distances.
- **Correct for this effect by using an effective refractive index, parameterised as function of photon propagation distance:**
  - Use linear parameterisation:

$$\boxed{\mathbf{n = n_0 + \alpha\ L_\gamma}}$$

  - Fit to 200kton WCSim data: n_0=1.33, α=4x10^{-4} m^{-1}.
- **I'm not sure how much of this effect is real dispersion.**

Timing Distributions

- **In likelihood function, the timing pdf $P_t(t)$ is constructed by parameterising the time residuals at the true vertex. The pdf contains the following components:**

  (1) Use a central Gaussian distribution.
  - For each hit, width of Gaussian is adjusted as a function of pulse height (according to parameterised resolution).

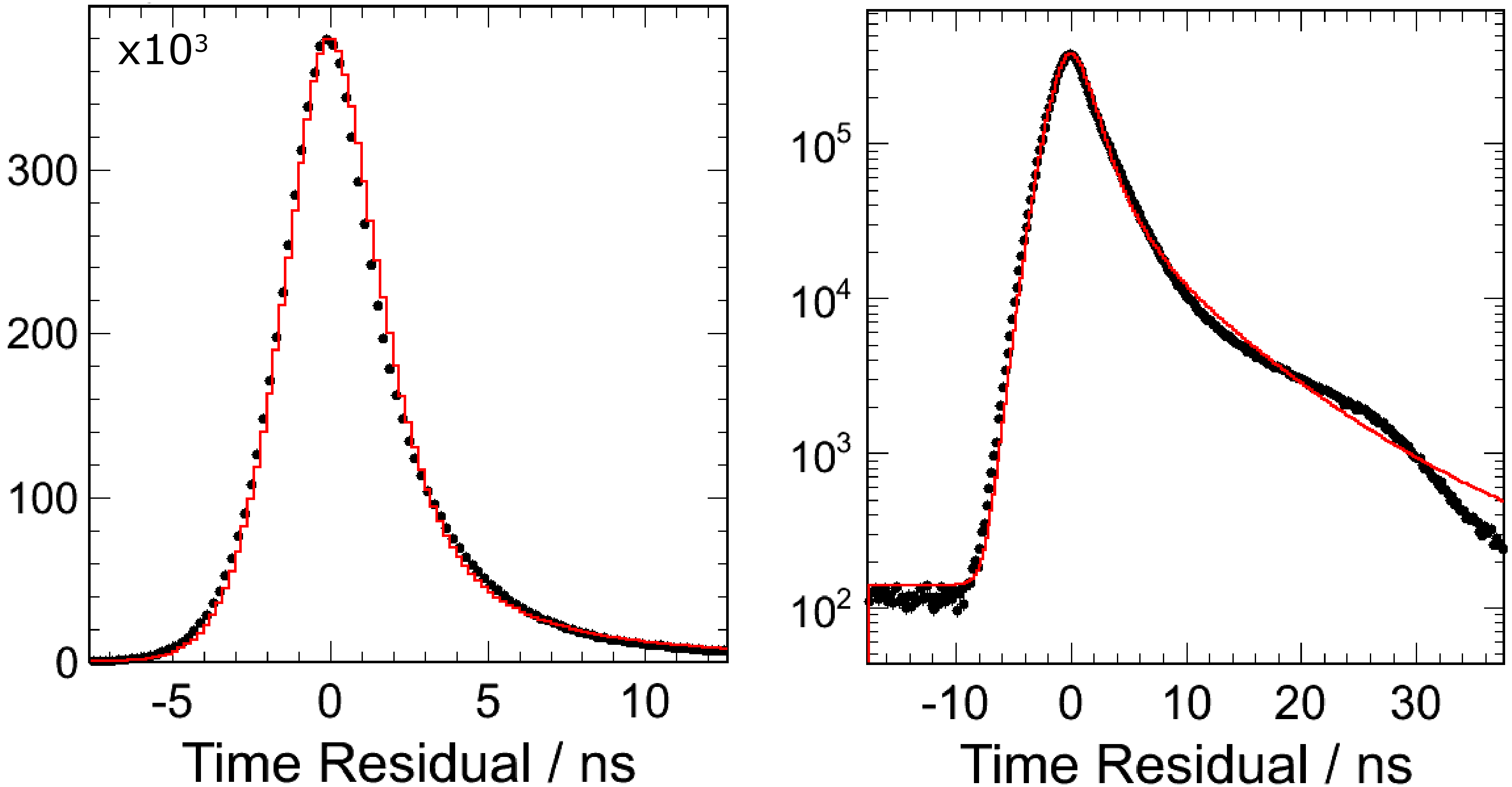
  (2) Add a double-exponential term to model late light.
  - Amplitude of term can be fitted as nuisance parameter for each event, but a constant fraction is used for now.

  (3) Add a small constant term for other noise.

Timing Distributions

- Distribution of timing residuals at true vertex (sum of μ and e).

— Sum of timing pdfs.



Angular Distribution

- **The spatial part of the likelihood function, $P_q(\theta)$, is constructed by parameterising the angular distribution of digits, assuming a spherical (!) geometry:**

$$P(\theta) = P_0 \sin(\theta) f(\theta \mid \alpha_0, \alpha_1, \beta)$$

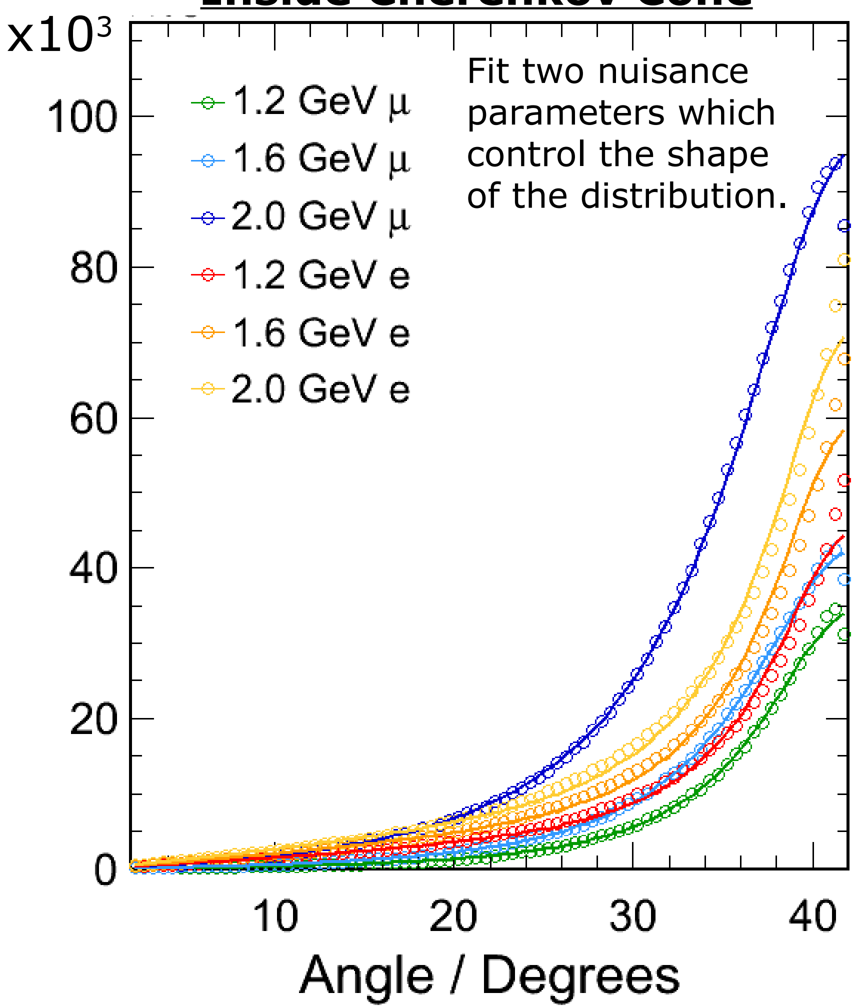
  - Here, $f(\theta)$ is a function that fits the angular distribution of events.

- **Inside Cherenkov cone (θ<42°):**
  - Take $f(\theta)$ to be a Lorentz function, with two shape parameters, which effectively fit the track length.
  - Fit these two shape parameters as nuisance parameters.

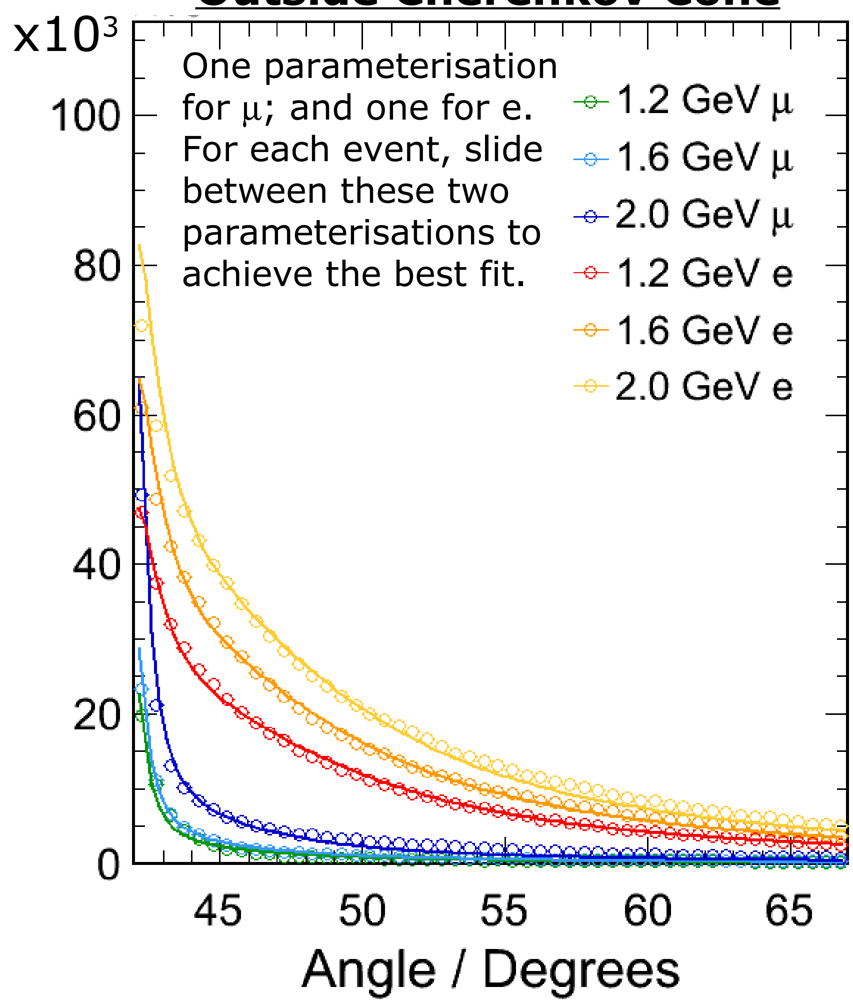
- **Outside Cherenkov cone (θ>42°):**
  - Parameterise the average distribution of muons and electrons.
  - Fit a linear combination of these two functions to each event, with a nuisance parameter, α, giving the relative amplitude.
    - ◇ Expect: $\alpha=0$ for electrons and $\alpha=1$ for muons.

Angular Distribution

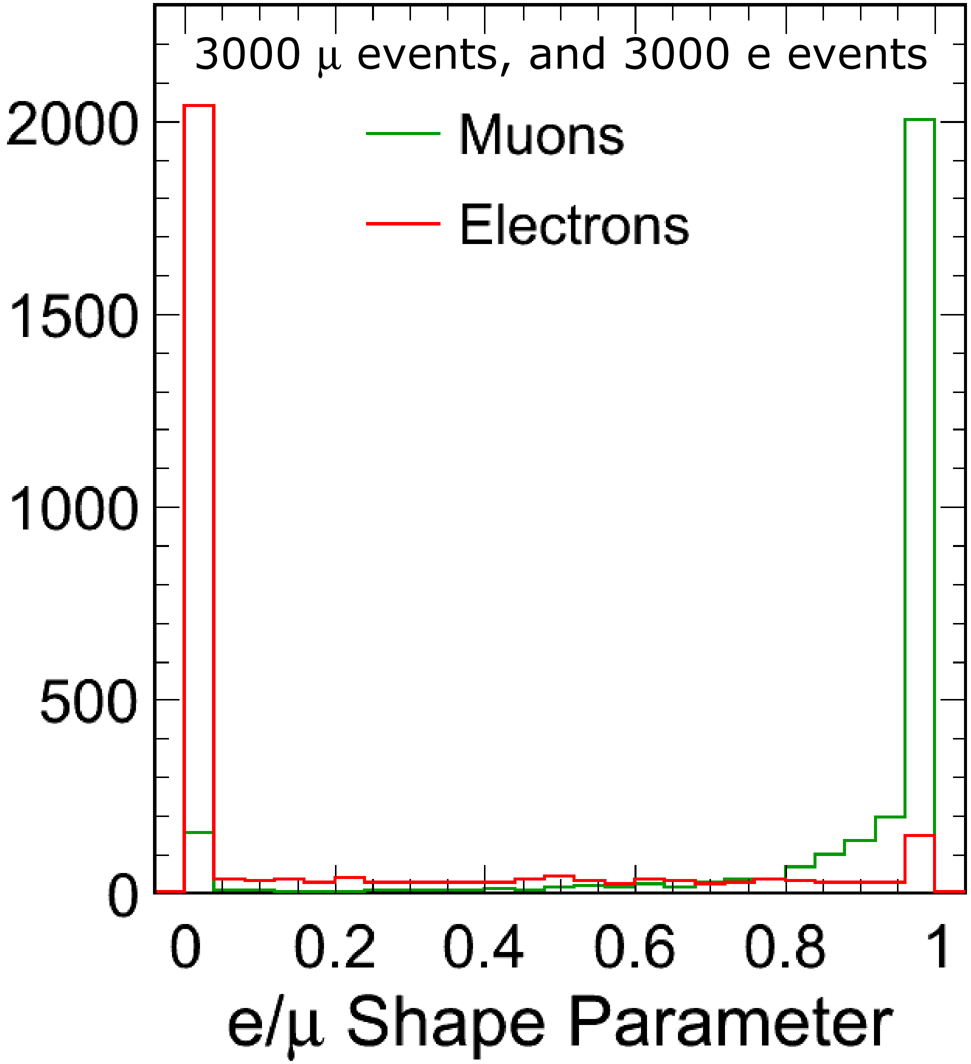
Inside Cherenkov Cone



Outside Cherenkov Cone



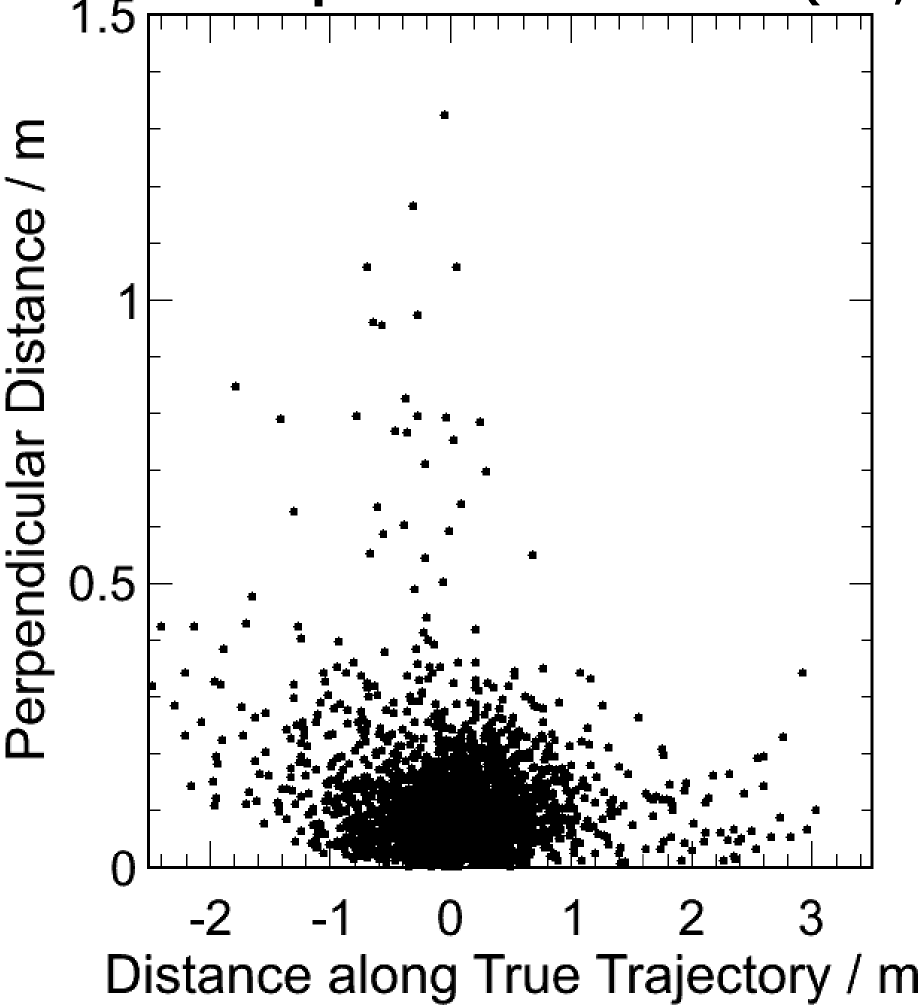
Electron/Muon Separation

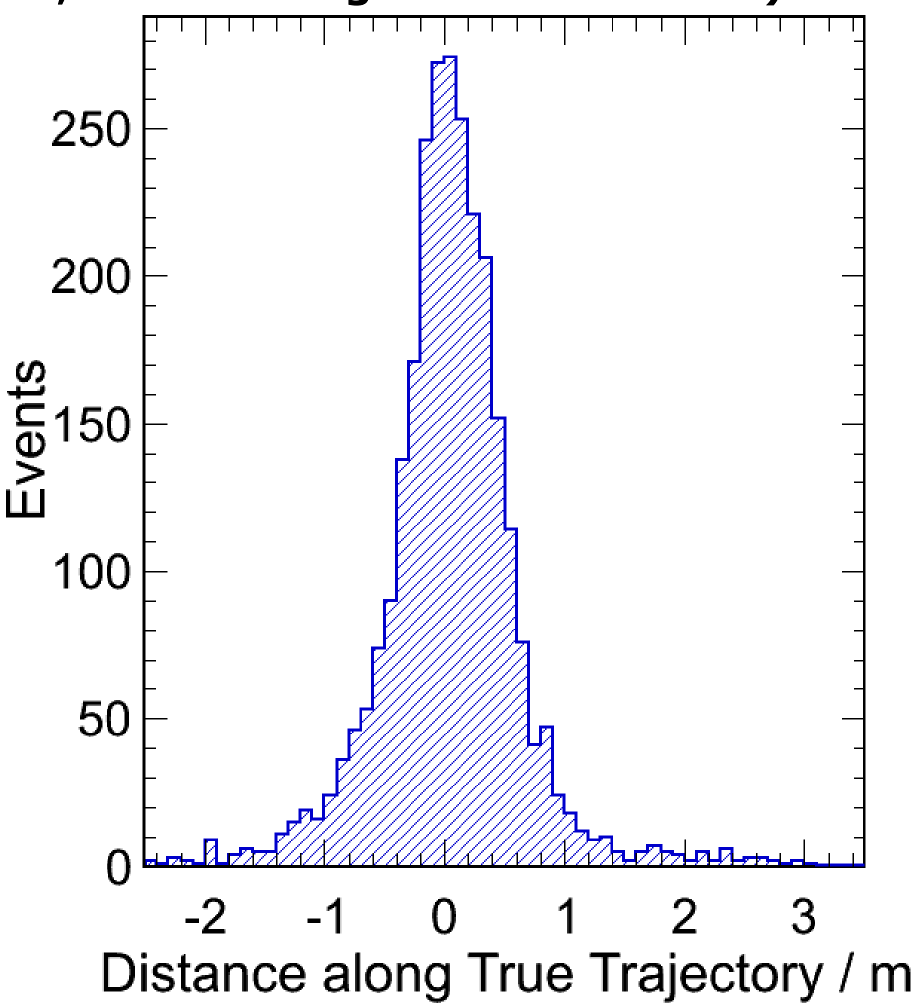


- **Outside Cherenkov cone:**
  - For hits outside 42° cone, the fit slides between the parameterised templates of muons and electrons.
    - ◇ A value of $\alpha=1$ implies that muon template is the best fit.
    - ◇ A value of $\alpha=0$ implies that electron template is the best fit.
  - Left plot shows that things work as they should!
    - ◇ For true μ, fitted values peak at $\alpha=1$.
    - ◇ For true e, fitted values peak at $\alpha=0$.
  - Peaks have ~93% purity.

Results of Vertex Fit: Muons

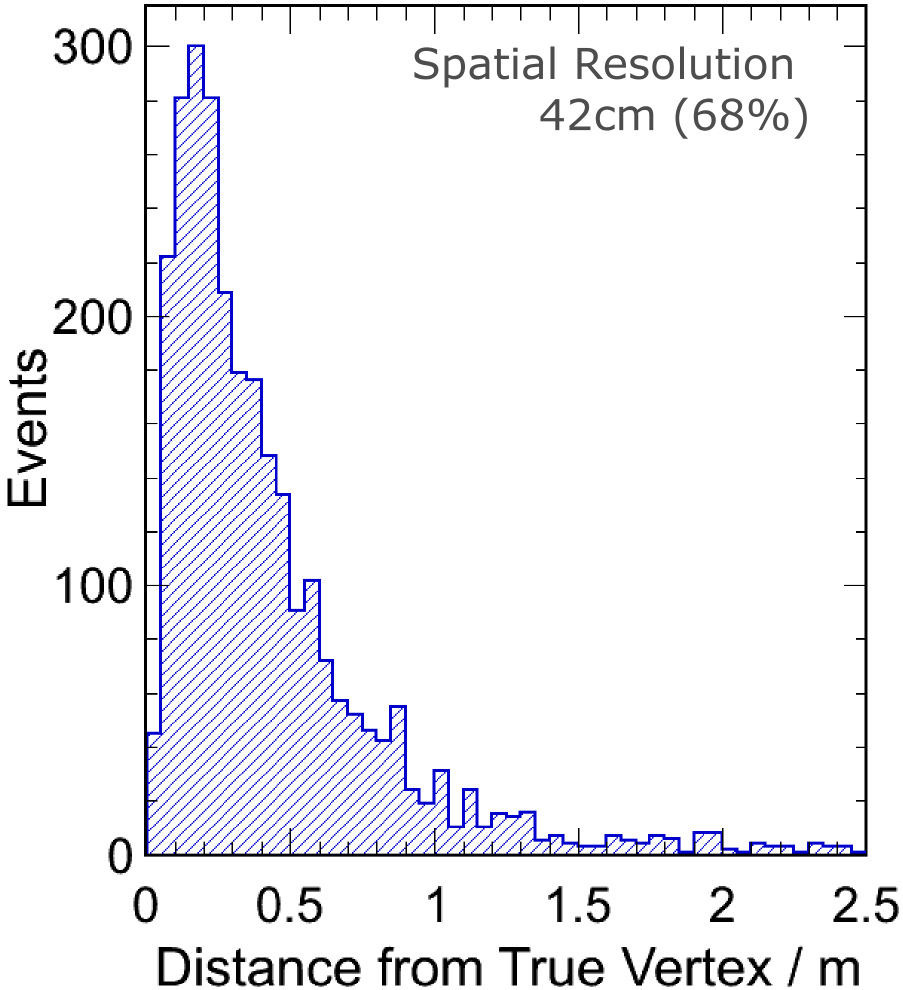
Sample of 3000 Events (1.2, 1.6, 2.0 GeV Single-Particle Muons).

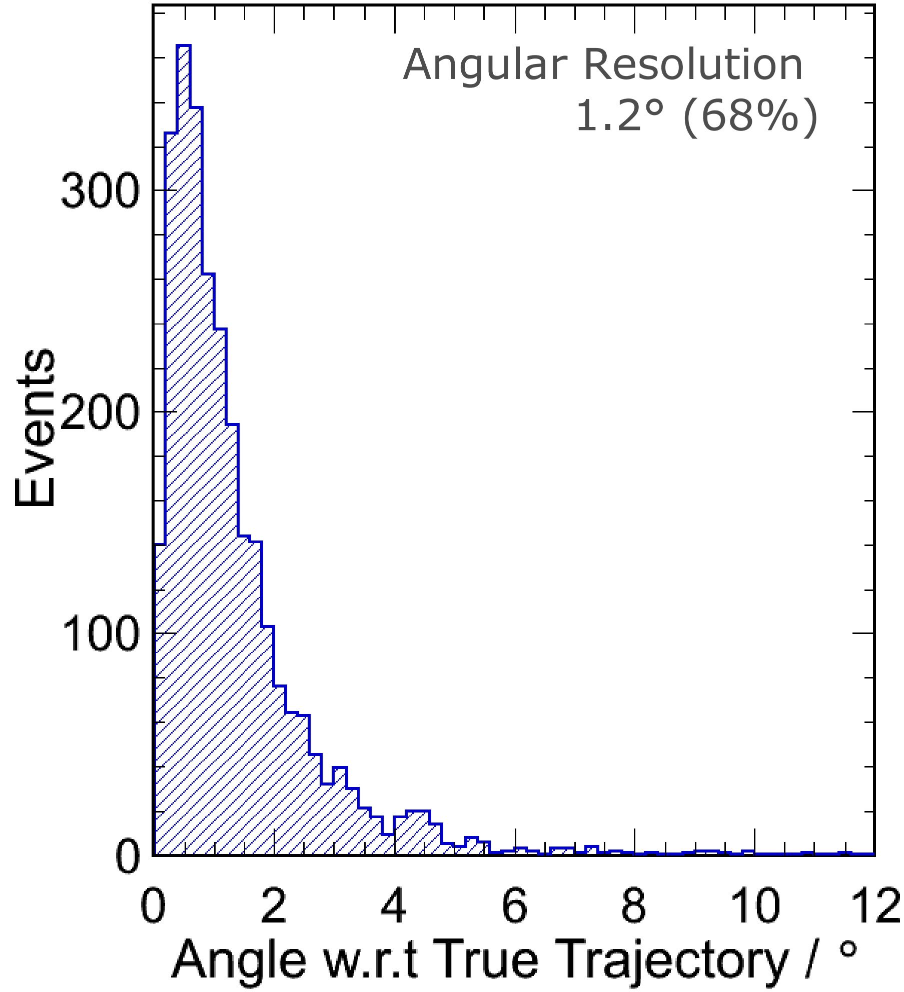




Results of Vertex Fit: Muons

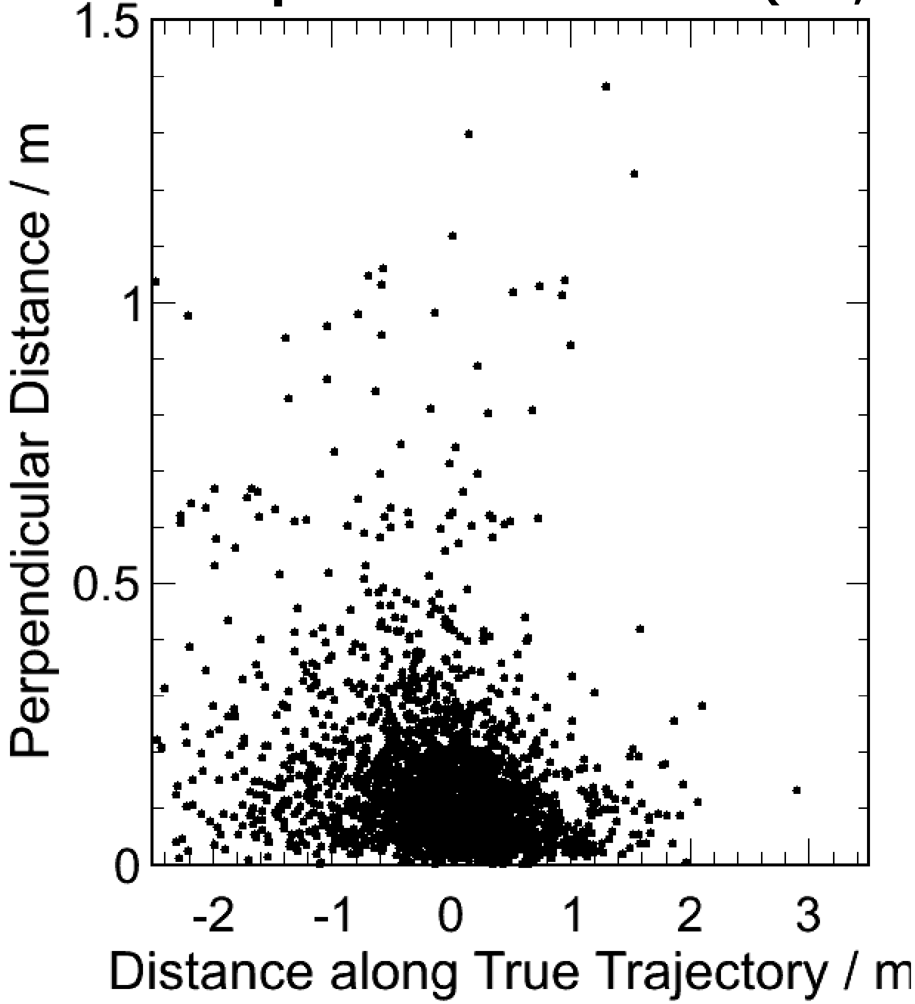
Sample of 3000 Events (1.2, 1.6, 2.0 GeV Single-Particle Muons).

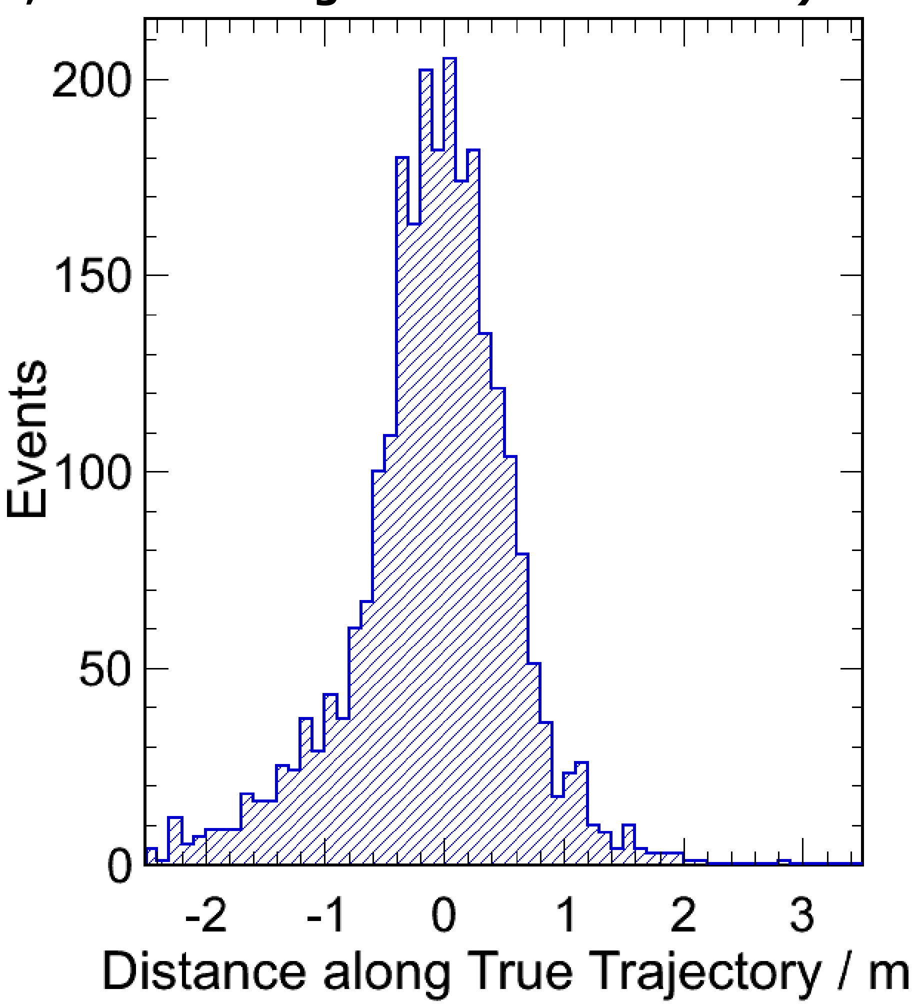




Results of Vertex Fit: Electrons

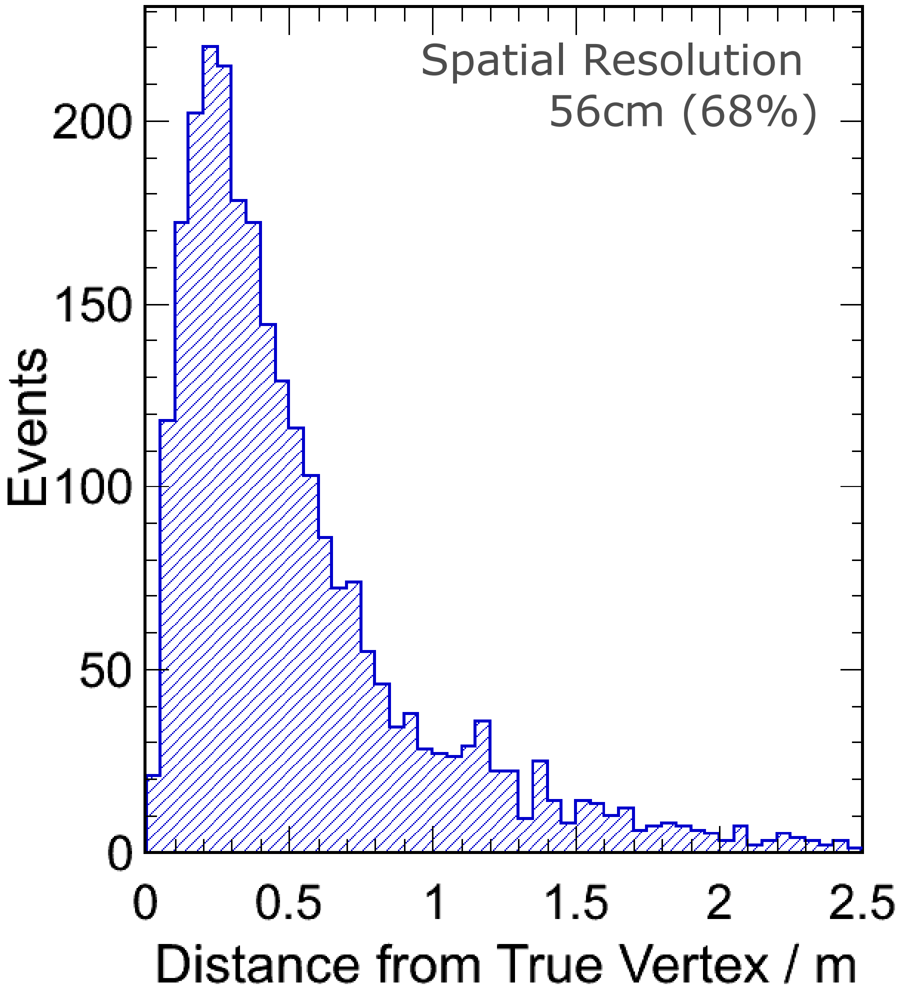
Sample of 3000 Events (1.2, 1.6, 2.0 GeV Single-Particle Electrons).

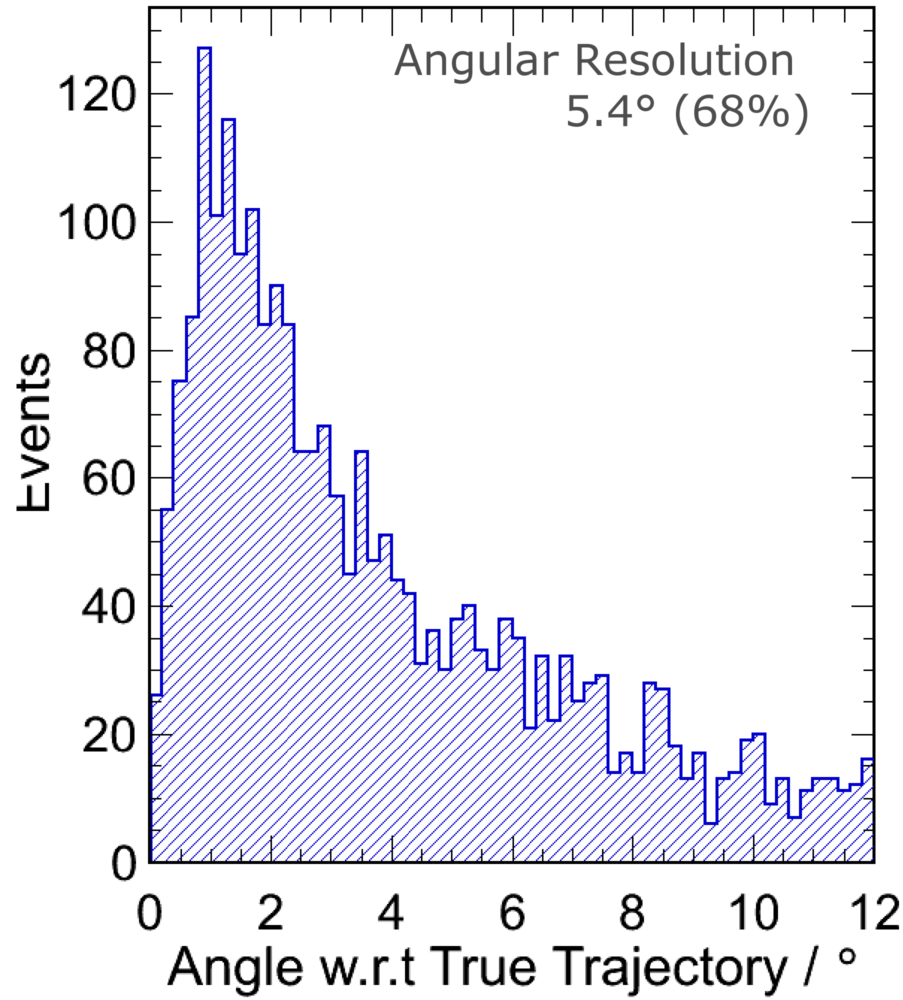




Results of Vertex Fit: Electrons

Sample of 3000 Events (1.2, 1.6, 2.0 GeV Single-Particle Electrons).

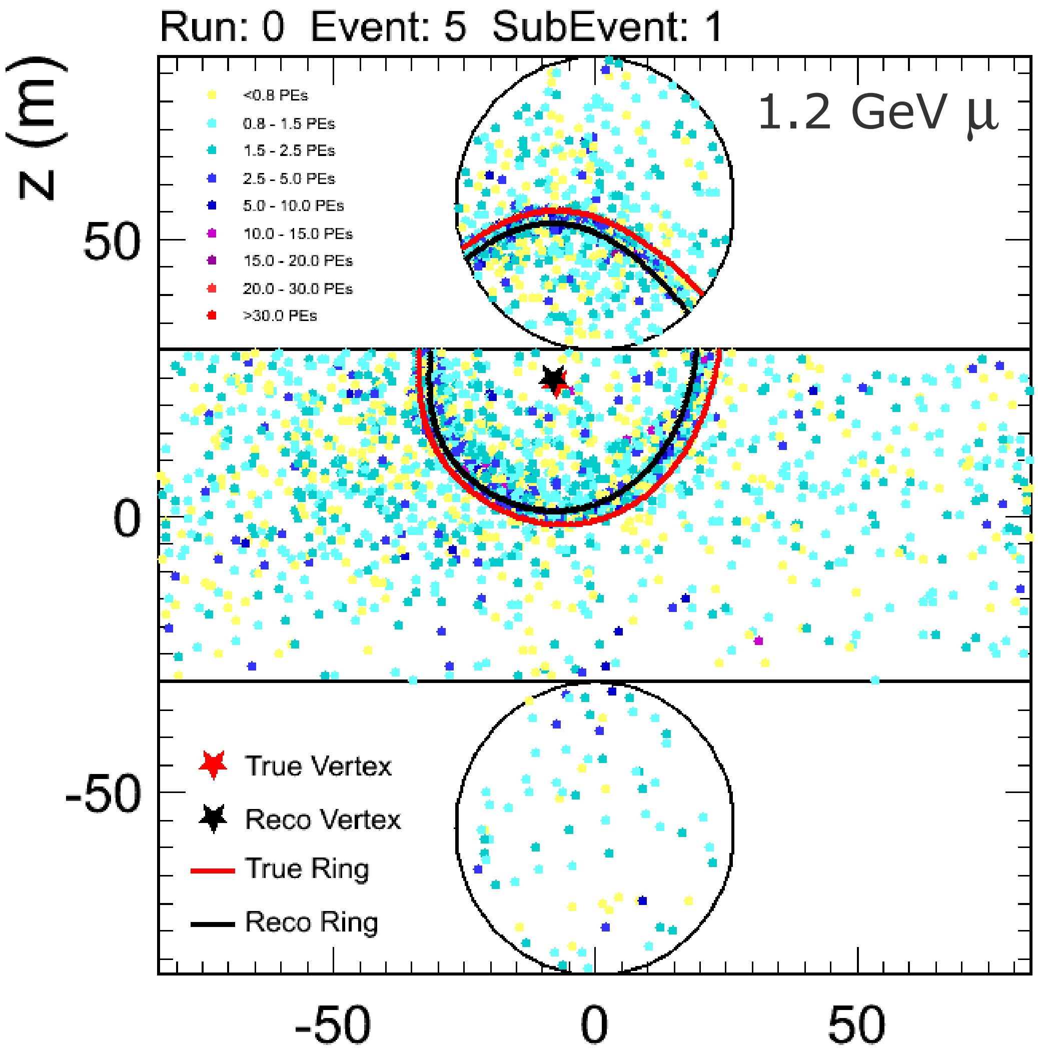


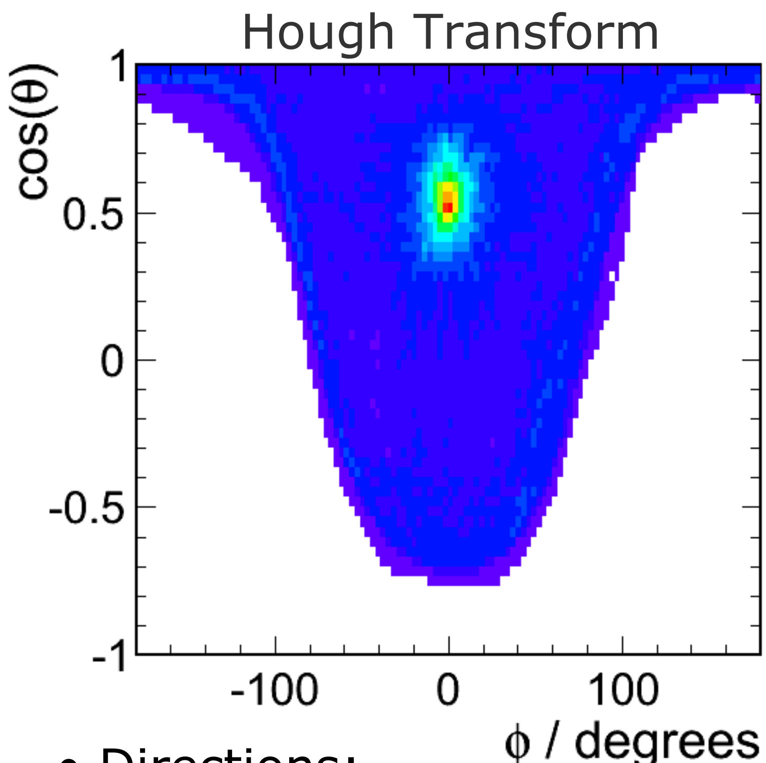


Single-Ring Finder

- **Have written a first pass ring finder. Algorithm is simple: apply Hough transform.**
  - Input reconstructed vertex.
  - Apply three-parameter Hough Transform.
    - ◇ Two direction angles (binning: 60 bins per 180 degrees).
    - ◇ Cherenkov cone angle (binning: 1 bin per degree).
  - Find position of Hough peak.
    - ◇ Take bin with maximum content.
    - ◇ Some refinement of peak position based on analysis of neighbouring bins.
  - Form a single ring.
    - ◇ Use direction and cone angle at Hough peak.
    - ◇ No further refinement of single ring.
    - ◇ No attempt to find second ring.
    - ◇ So, needs more work!

Example Event





- Directions:
  θ = zenith angle.
  φ = azimuthal angle.
- Hough transform histogram shown at the cone angle with the largest peak.

Note: ‘true’ ring drawn assuming fixed cone angle of 42 degrees.

Summary

- **A framework has been developed for WC reconstruction tools.**
  - Reads in the WCSim ROOT files and provides handles to digits, detector geometry, and Monte Carlo truth information.
  - Also home to a couple of event displays.
  - Contains first-pass set of data classes, along with a framework for running reconstruction algorithms.
- **Also provided are a set of simple reconstruction algorithms.**
  - Some clustering agorithms for 'cleaning' events, if necessary.
  - Vertex fitter, now tuned for the 200 kton detector geometry.
    - ◇ Uses Minuit to minimise 9-parameter likelihood function.
    - ◇ Gives vertex resolution of ~50cm for single-particle μ and e, along with separation of μ and e with >90% purity.
    - ◇ Not a full fit. Doesn't calculate full hit-by-hit expectations.
    - ◇ Instead, analyses hits using global pdfs of timing residuals and angular distributions.
  - Three-parameter Hough transform, used to find single rings.
- **Code is in SVN, and provides base for CommonReconstruction.**