

LArSoft Recombination Modelling

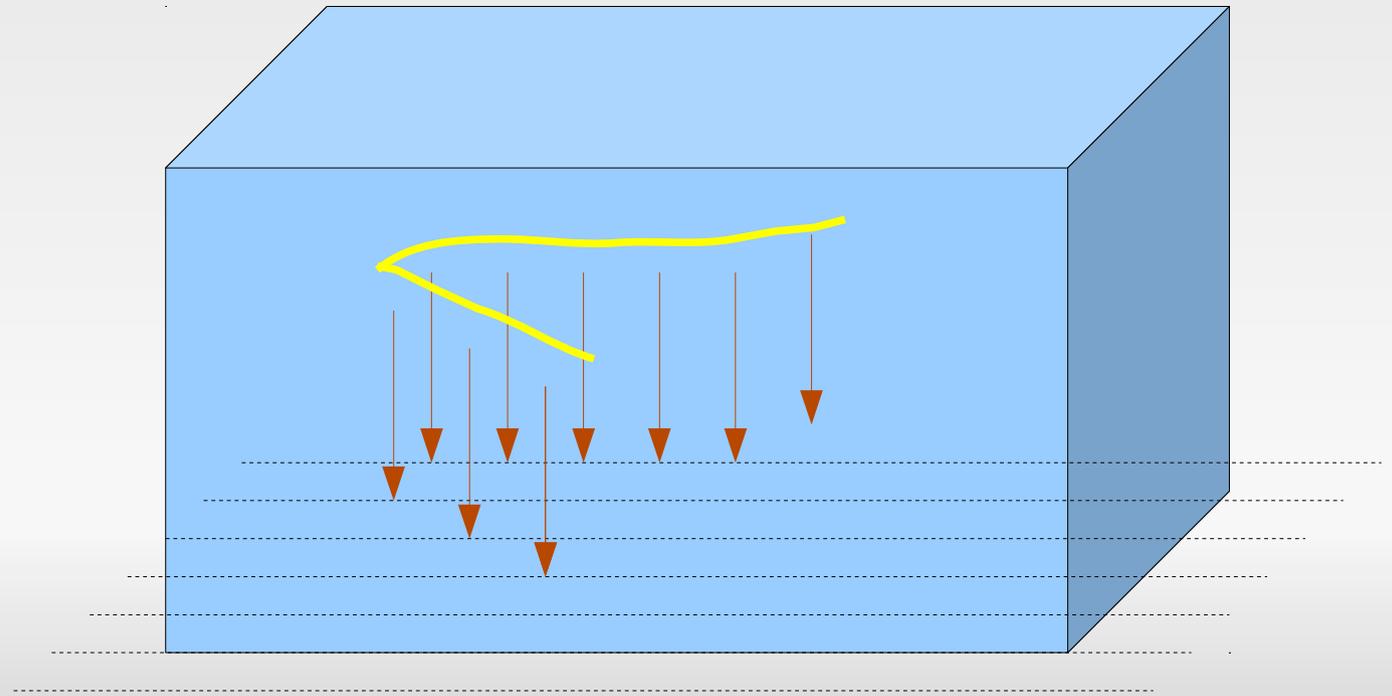
Ben Jones, MIT

Setting the Scene

DriftElectrons Module:

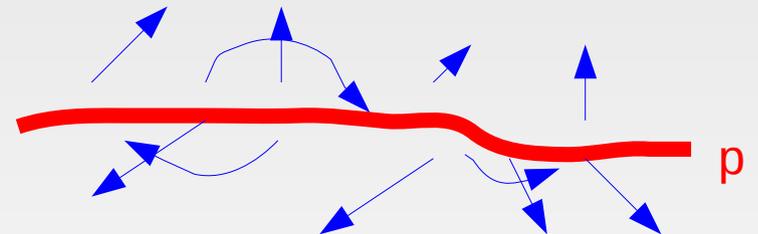
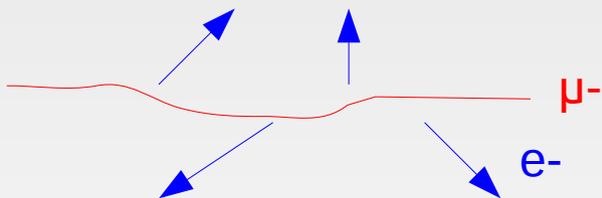
Simple simulation module which:

- calculates charge deposit per voxel based on energy losses of particles in LArG4
- models charge losses / quenching due to recombination effects
- models loss or diffusion of charge along drift direction
- determines charge arriving at each wire in wire planes



Recombination

- LArG4 outputs the total energy lost by particles traversing each voxel in the event. These losses are assumed to be due to ionization deposits.
- Liberating an ionization electron costs some amount of energy – at low energies and charge densities, the relationship between number of electrons freed and the energy loss by an ionizing particle is ~linear.
- For higher energy losses, the track core has a large positive charge density, and so recombination of the freed electrons can take place. This leads to an asymptotic charge density per track length at high dE/dx



- The relationship between the energy lost along some small track length, and the ionization charge liberated, is well modelled by the relationship below, with $A \sim 0.8$ and $k \sim 0.097 \text{ cm} / \text{MeV}$

$$Q(x) = \delta E(x) \left[\frac{A}{1 + k \frac{dE}{dx}} \right]$$

Recom in DriftElectrons

LArSoft-SVN

[Overview](#) [Activity](#) [Issues](#) [Gantt](#) [Calendar](#) [News](#) [Documents](#) [Wiki](#) [Files](#) [Repository](#)

Bug #1095

Incorrect energy deposit handling by DriftElectrons

Added by Benjamin Jones 20 days ago. Updated 18 days ago.

Status:	New	Start:	30/03/2011
Priority:	Normal	Due date:	
Assigned to:	-	% Done:	<input type="text"/> 0%
Category:	-	Spent time:	-
Target version:	-		

Description

In the DriftElectrons module, the charge yield for a voxel gets adjusted for recombination effects by a factor of

```
double recomb = fRecombA/(1. + Energy*(1e3/lvc->VoxelSizeX())*fRecombk)
```

However, VoxelSizeX() is the voxel width, not the track length. The actual track length within this voxel is something in between 0 and $\sqrt{X^2 + Y^2 + Z^2}$. Hence this simulation method miscalculates the energy deposits in each voxel after recombination.

This is especially bad for the case where voxels are non-isotropic, since most particles will be moving at least somewhat in the Z direction rather than the X direction. The charge reported per voxel will be systematically more wrong for tracks at higher track angles, and the problem is most significant for particles with large dE/dx where the second term in the denominator dominates.

Not clear the best way to rectify this - either account for recombination effects in LArG4 before storing the LArVoxelList or store some information about total accumulated track length in each voxel.

Recom in DriftElectrons

History

Updated by William Seligman 18 days ago

#1

I spoke with Mike Shaevitz about this issue. Any physics insight given below is his; any mistakes are mine.

The issue is how much do we want to "pay" to do this calculation right.

It's quite possible for more than one particle to pass through a voxel. The way DriftElectrons works now, it just takes the total energy deposited by ($N \geq 1$) particles and divides it by the width of the voxel. This is an approximation with some error.

We could do better. Right now, for each voxel I also have a list of particles that contributed to the voxel energy (above a cut given in a .fcl file). If one knew that different types of particles had different ionization amounts, one could separately correct for each one.

But as Ben points out, what you need is the track length within the voxel. This is something that can be determined in the Monte Carlo; in addition to writing the total energy in the voxel, I could also write the sum of all the track lengths in the voxel. To be even more accurate, I could add the track length of each particle in the per-voxel particle list. Then one could know the actual dE/dx of each particle in the voxel, and even include saturation effects if that became relevant.

This would:

- Allow a better approximation for recombination effects;
- Increase the amount of memory used in the voxel list;
- Increase the computation time within LArG4.

Now I have to bounce the question(s) back: How much of an error are we making with the current approximation, and how much would it be improved by any of the above steps, and is that improvement worth the additional computation costs as opposed to folding it into systematic error?

Someone could do a study to answer these questions. It doesn't have to be too sophisticated; a simple model of a voxel with different particles going through at different angles. Sounds like a nice grad-student project to me.

Recom in DriftElectrons

History

Updated by William Seligman 18 days ago

#1

I spoke with Mike Shaevitz about this issue. Any physics insight given below is his; any mistakes are mine.

The issue is how much do we want to "pay" to do this calculation right.

It's quite possible for more than one particle to pass through a voxel. The way DriftElectrons works now, it just takes the total energy deposited by ($N \geq 1$) particles and divides it by the width of the voxel. This is an approximation with some error.

We could do better. Right now, for each voxel I also have a list of particles that contributed to the voxel energy (above a cut given in a .fcl file). If one knew that different types of particles had different ionization amounts, one could separately correct for each one.

But as Ben points out, what you need is the track length within the voxel. This is something that can be determined in the Monte Carlo; in addition to writing the total energy in the voxel, I could also write the sum of all the track lengths in the voxel. To be even more accurate, I could add the track length of each particle in the per-voxel particle list. Then one could know the actual dE/dx of each particle in the voxel, and even include saturation effects if that became relevant.

This would:

- Allow a better approximation for recombination effects;
- Increase the amount of memory used in the voxel list;
- Increase the computation time within LArG4.

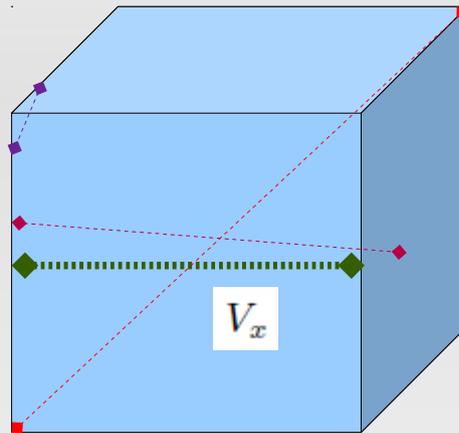
Now I have to bounce the question(s) back: How much of an error are we making with the current approximation, and how much would it be improved by any of the above steps, and is that improvement worth the additional computation costs as opposed to folding it into systematic error?

Someone could do a study to answer these questions. It doesn't have to be too sophisticated; a simple model of a voxel with different particles going through at different angles. Sounds like a nice grad-student project to me.

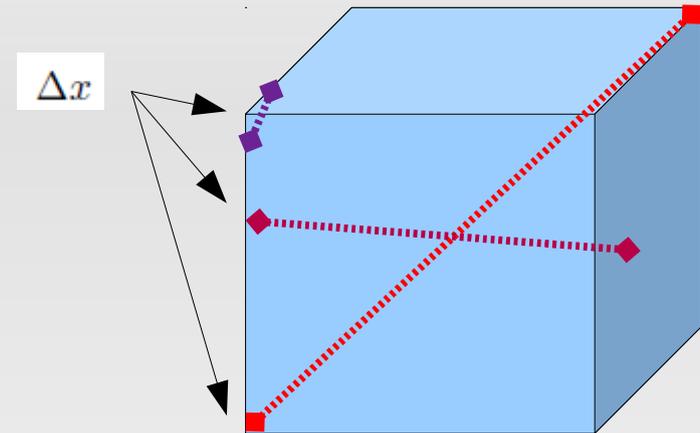
The Problem

Track
Geometry

Current Implementation



True Geometry



Assumed Track
Length

V_x

Δx

Energy in Voxel
from track

$$E = \frac{dE}{dx} \Delta x$$

$$E = \frac{dE}{dx} \Delta x$$

Charge
calculated

$$Q_{Approx} = \frac{A}{1 + k \left(\frac{\Delta x}{V_x} \right) \frac{dE}{dx}} \cdot \frac{dE}{dx} \Delta x$$

$$Q_{True} = \frac{A}{1 + k \frac{dE}{dx}} \cdot \frac{dE}{dx} \Delta x$$

Currently used in LArSoft

Mismeasurement at different DeltaX Values

How badly the LarSoft approximation mismeasures the charge per voxel depends on several different things:

- 1) dE/dx of track
- 2) Voxel size
- 3) Voxel geometry (isotropic / nonisotropic)

Also, most importantly, the discrepancy in one voxel is a function of DeltaX (track length in voxel). For standard LarSoft voxels, we can look at the charge measured per voxel at different DeltaX values using the approximate and true length calculations:

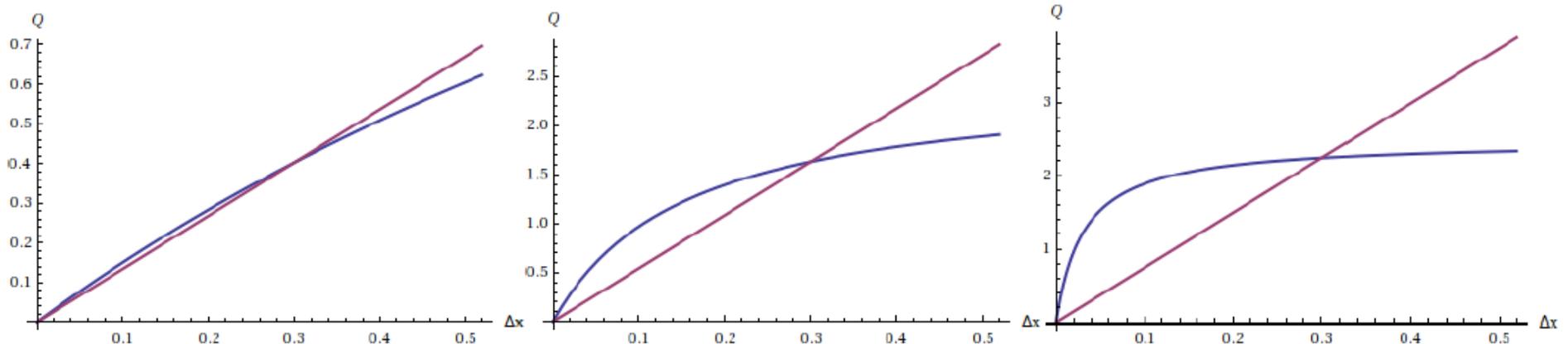


Fig. 3: Charge reported by the larsoft voxel length approximation (blue) and true length expression (purple) for tracks of constant $dE/dx = 2 \text{ MeV/cm}$ (left), 20 MeV/cm (centre), 100 MeV/cm (right).

Mismeasurement at different DeltaX Values

$$Q_{Approx} = \frac{A}{1 + k \left(\frac{\Delta x}{V_X} \right) \frac{dE}{dx}} \cdot \frac{dE}{dx} \Delta x$$

$$Q_{True} = \frac{A}{1 + k \frac{dE}{dx}} \cdot \frac{dE}{dx} \Delta x$$

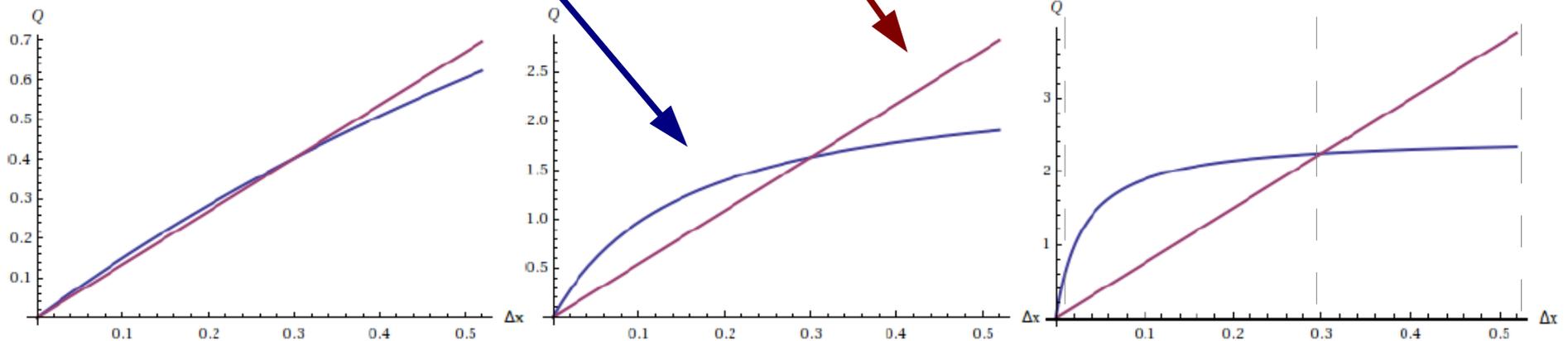
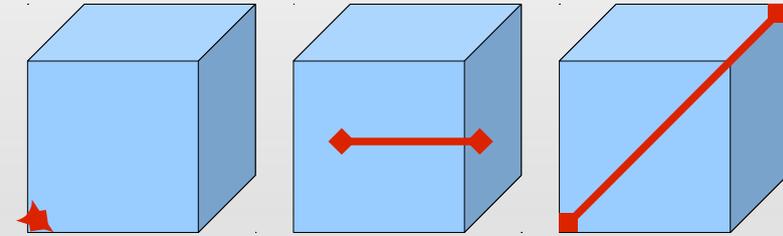
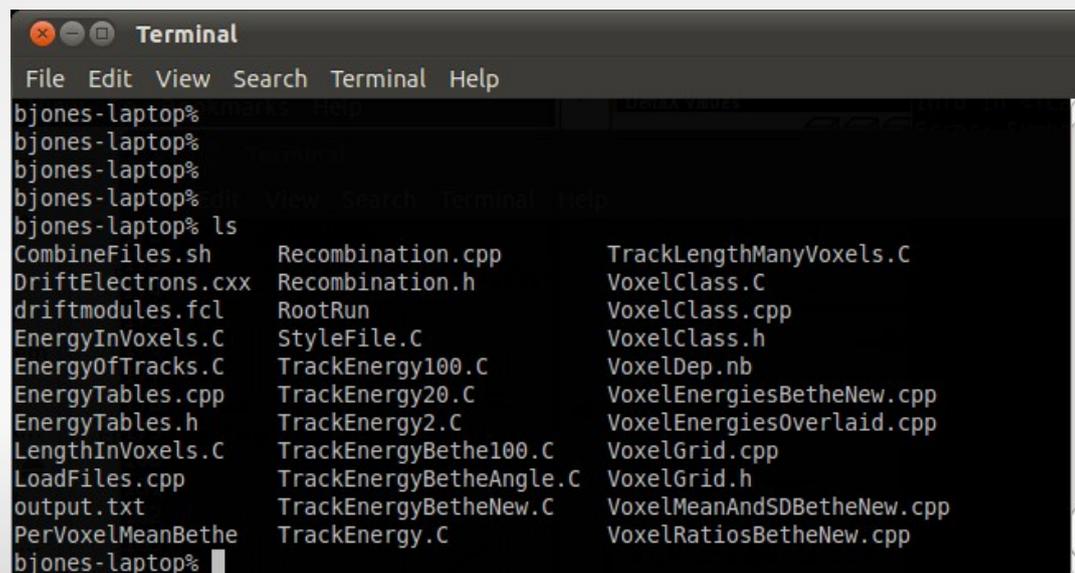


Fig. 3: Charge reported by the larsoft voxel length approximation (blue) and true length expression (purple) for tracks of constant $dE/dx = 2 \text{ MeV/cm}$ (left), 20 MeV/cm (centre), 100 MeV/cm (right).

Toy Monte Carlo

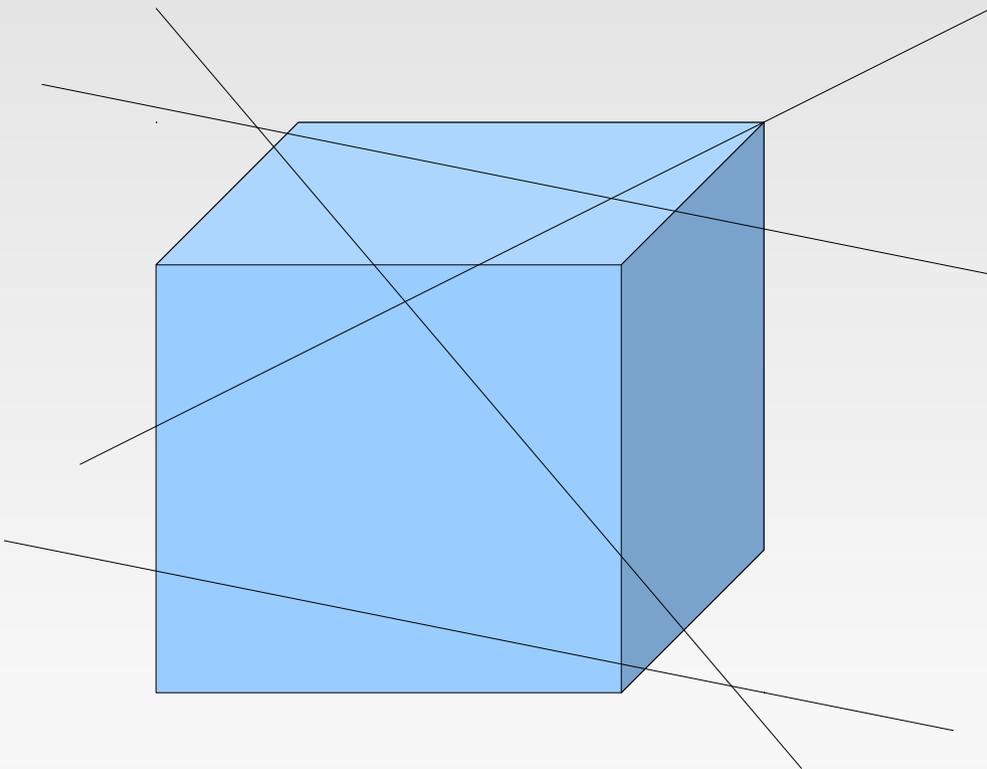
- Plots on previous page were analytical results for one voxel.
- Now move to tracks penetrating multiple voxels using a home-brewed C++ monte carlo.
- This assumes straight line tracks, which can be set to either constant dE/dx or Bethe Bloch energy losses, moving through a voxelized space, and records energy losses and charges from the true length and voxel width approximation, per voxel.
- These results are simplifications, and this MC is only intended to give an idea of the scale of the problem and assess the best direction to pursue. The real systematic error in track energy resolution needs to be assessed in LarSoft once a solution is chosen.



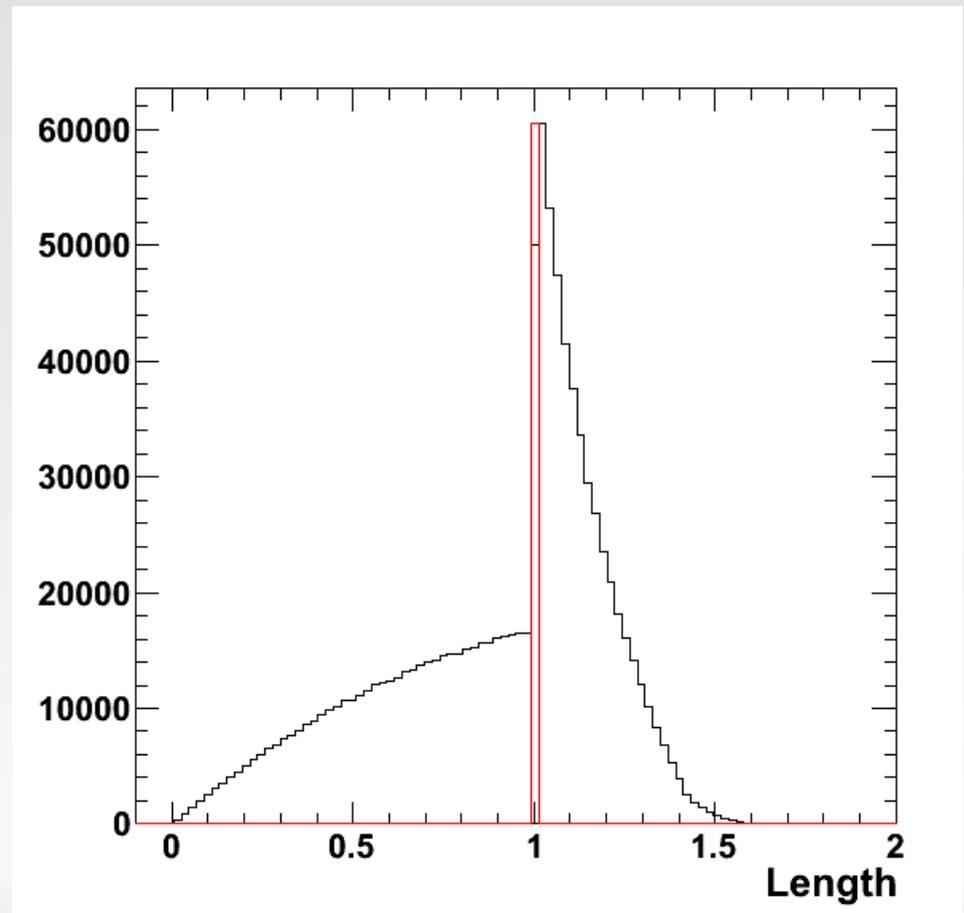
```
Terminal
File Edit View Search Terminal Help
bjones-laptop%
bjones-laptop%
bjones-laptop%
bjones-laptop%
bjones-laptop% ls
CombineFiles.sh      Recombination.cpp      TrackLengthManyVoxels.C
DriftElectrons.cxx  Recombination.h        VoxelClass.C
driftmodules.fcl    RootRun                 VoxelClass.cpp
EnergyInVoxels.C    StyleFile.C            VoxelClass.h
EnergyOfTracks.C    TrackEnergy100.C       VoxelDep.nb
EnergyTables.cpp    TrackEnergy20.C        VoxelEnergiesBetheNew.cpp
EnergyTables.h      TrackEnergy2.C         VoxelEnergiesOverlaid.cpp
LengthInVoxels.C    TrackEnergyBethe100.C  VoxelGrid.cpp
LoadFiles.cpp       TrackEnergyBetheAngle.C VoxelGrid.h
output.txt          TrackEnergyBetheNew.C  VoxelMeanAndSDBetheNew.cpp
PerVoxelMeanBethe  TrackEnergy.C          VoxelRatiosBetheNew.cpp
bjones-laptop%
```

Simple Geometry Investigation

- Luckily, the majority of tracks through a cuboidal voxel have DeltaX near to 1, so perhaps it isn't as bad as it looks.
- For random tracks, how are the DeltaX values distributed?



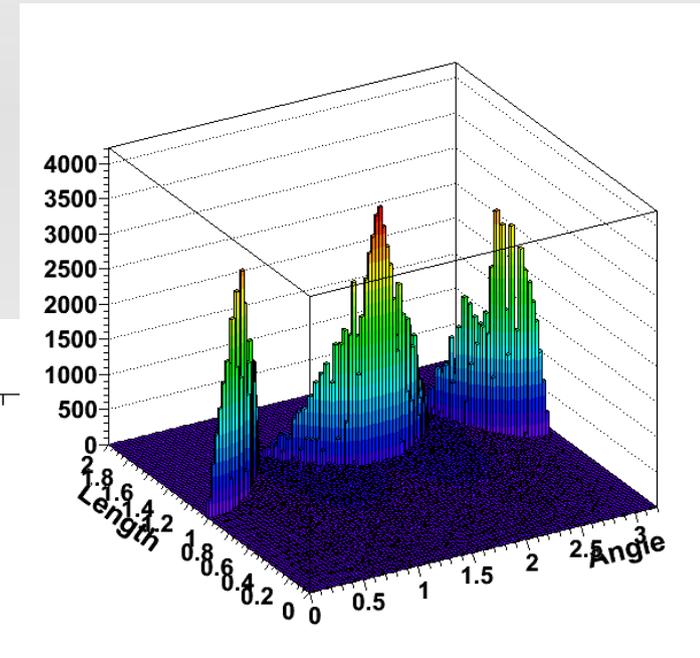
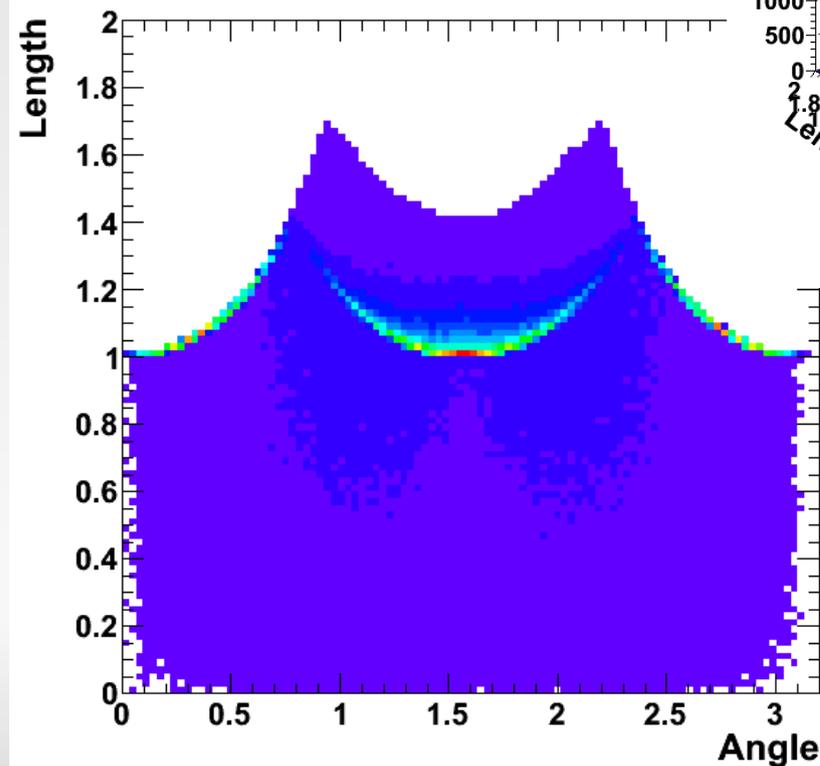
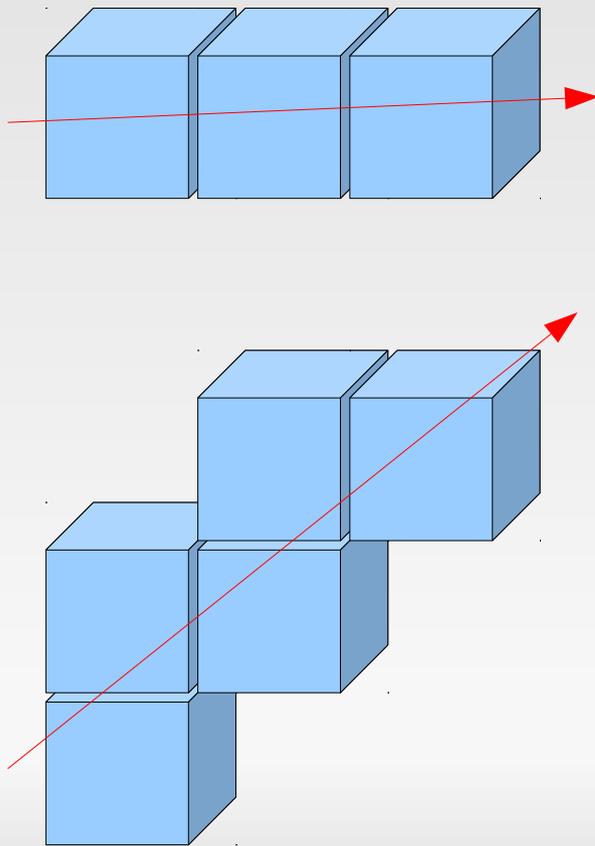
1 x 1 x 1 voxel



Angular Dependence

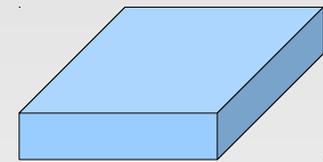
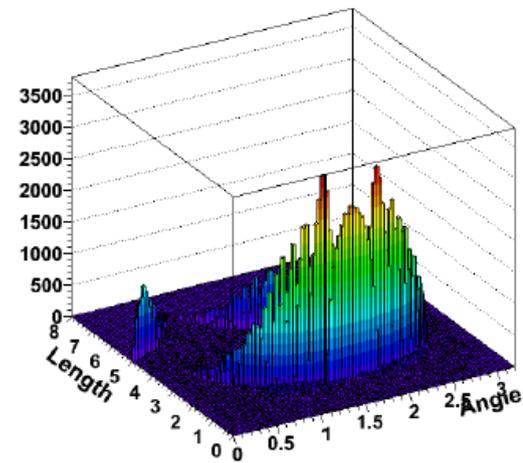
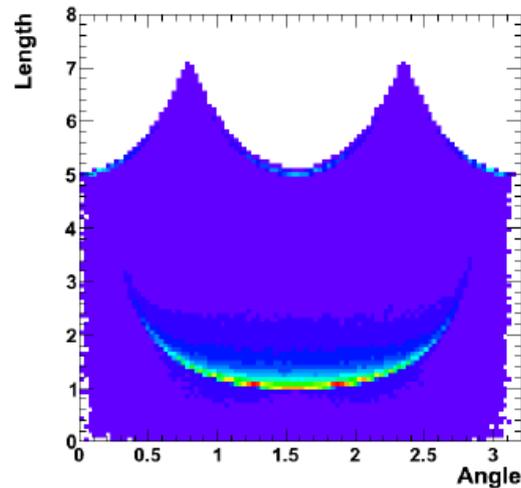
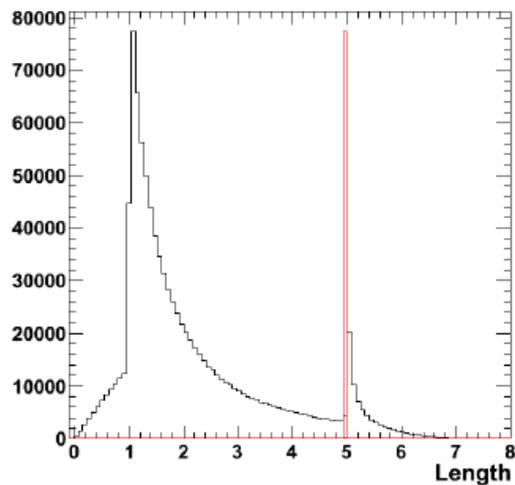
- Higher angle tracks will in general have a bigger discrepancy per voxel... Also interesting to look at DeltaX distribution as a function of angle

1 x 1 x 1 voxels

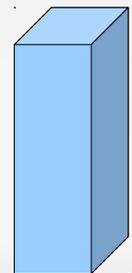
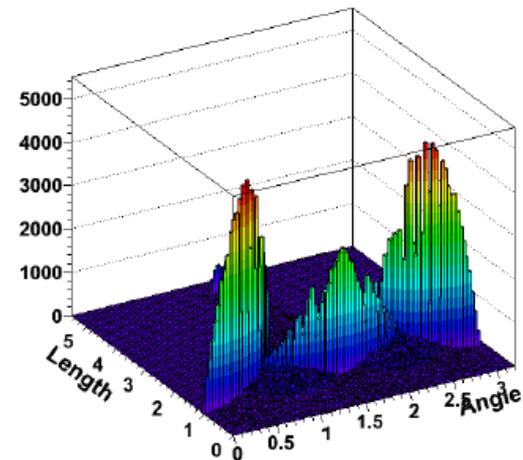
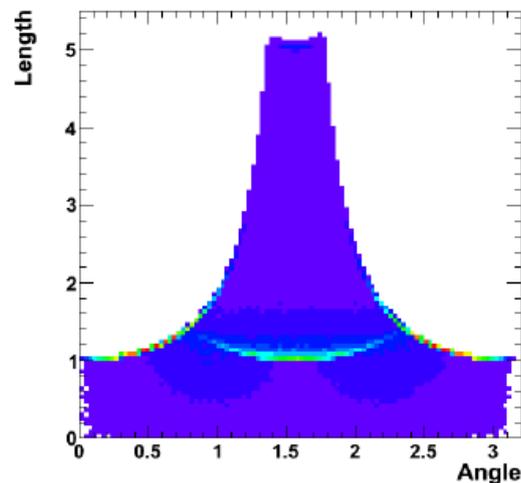
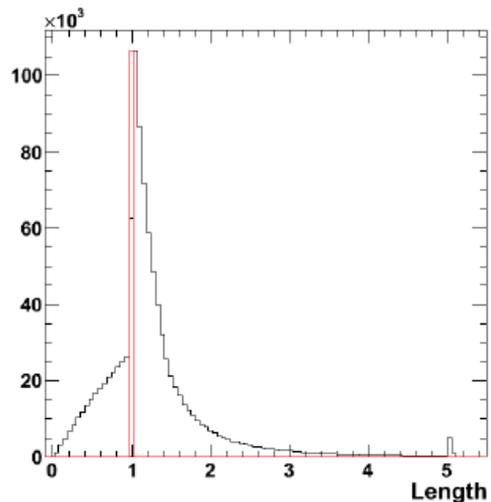


Nonisotropic Voxels (1)

In theory, LarSoft supports nonisotropic voxels. In practice, even if we can get away with the voxel length approximation for isotropic case, oblong voxels will totally ruin the charge measurement.



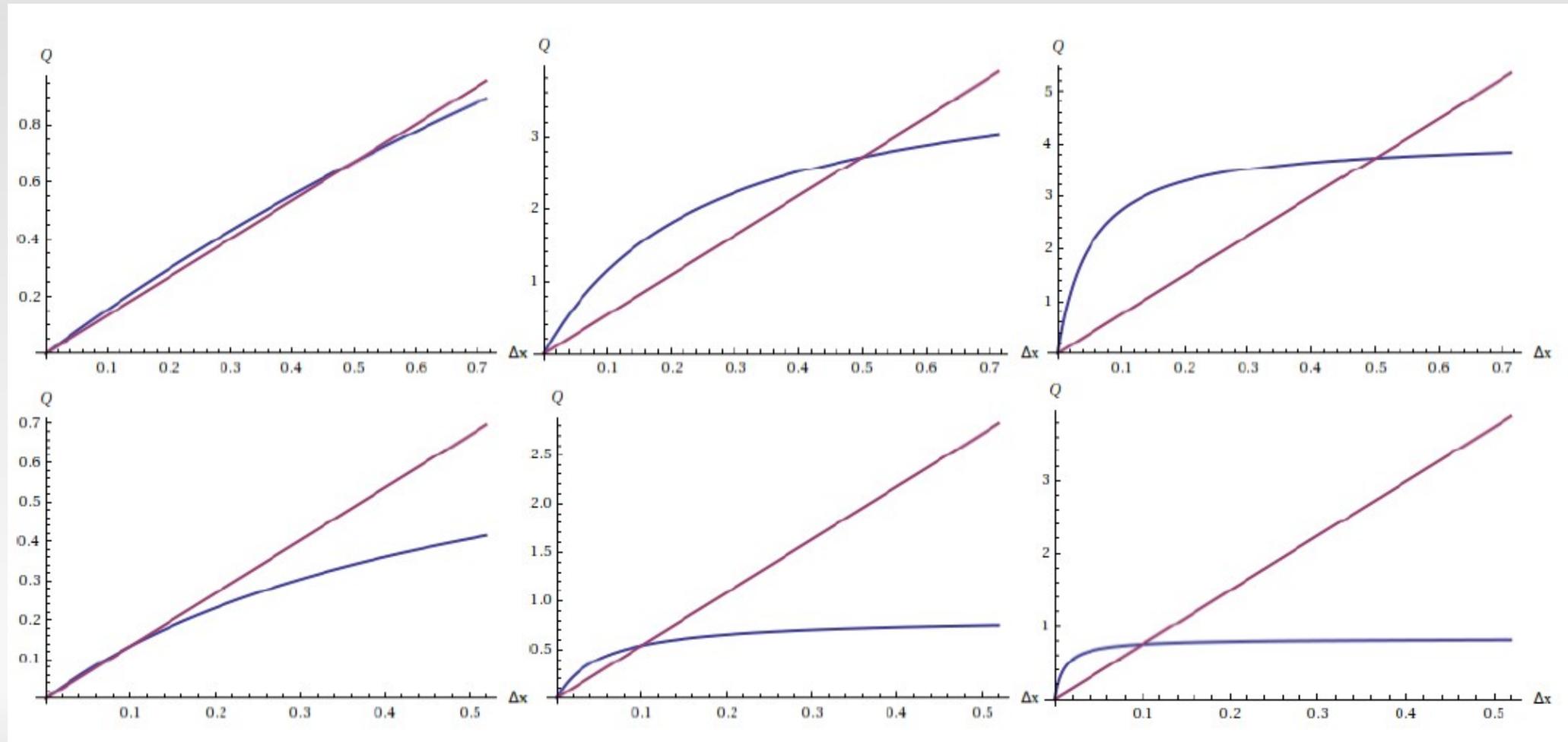
$5 \times 5 \times 1$



$1 \times 1 \times 5$

Nonisotropic Voxels (2)

In theory, LarSoft supports nonisotropic voxels. In practice, even if we can get away with the voxel length approximation for isotropic case, oblong voxels will totally ruin the charge measurement.

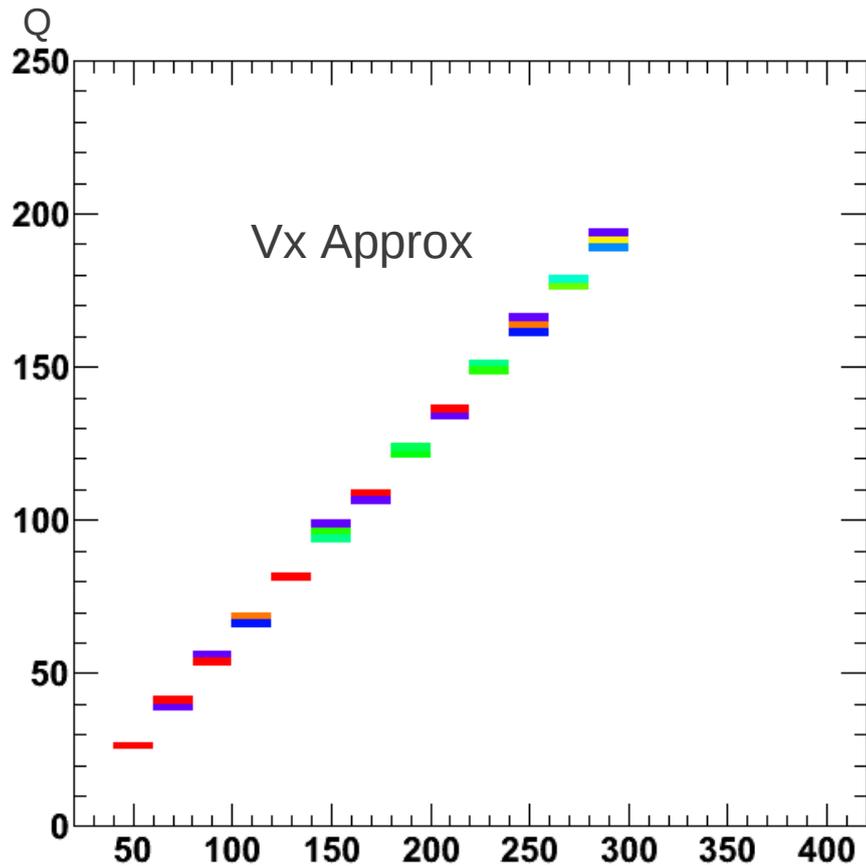


Tracks

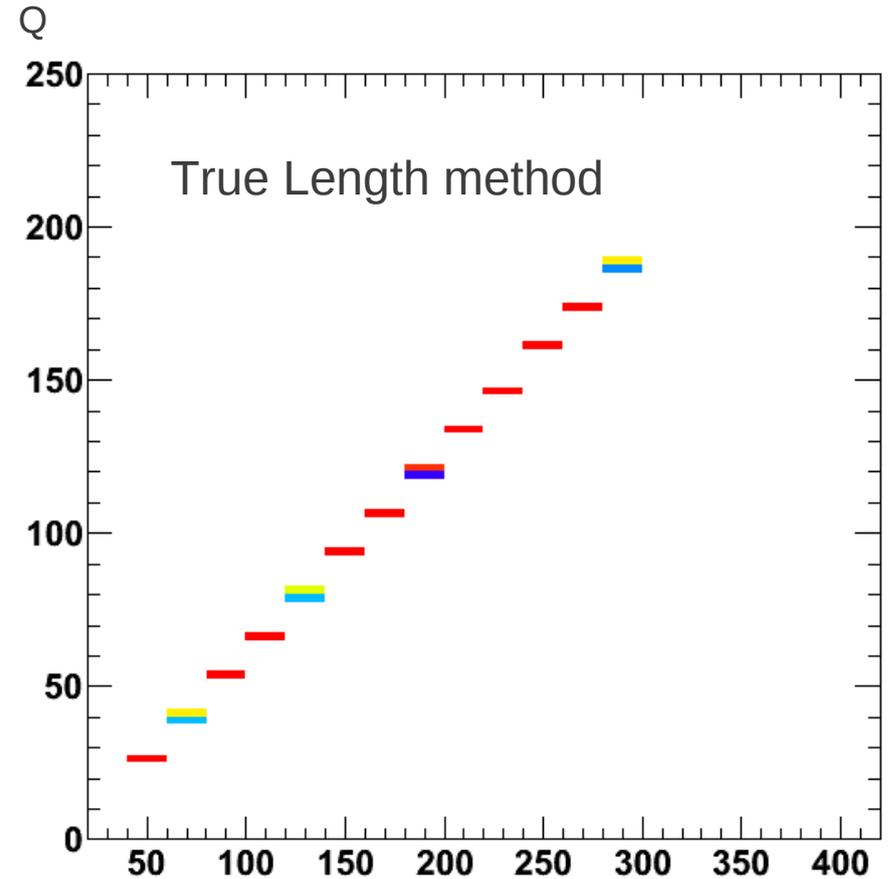
- Energy deposits in isolated voxels are not very interesting. What we really care about are the collective voxels from a track.
- Start by generating tracks of constant dE/dx at several reasonable values. Then we can ask what total charge we measure by summing the charge in all voxels. If we are lucky, some voxels will fluctuate high and others will fluctuate low, leading to a reasonable agreement between the two methods.

$dE/dx = 2 \text{ MeV / cm}$

$dE/dx = 20 \text{ MeV / cm}$

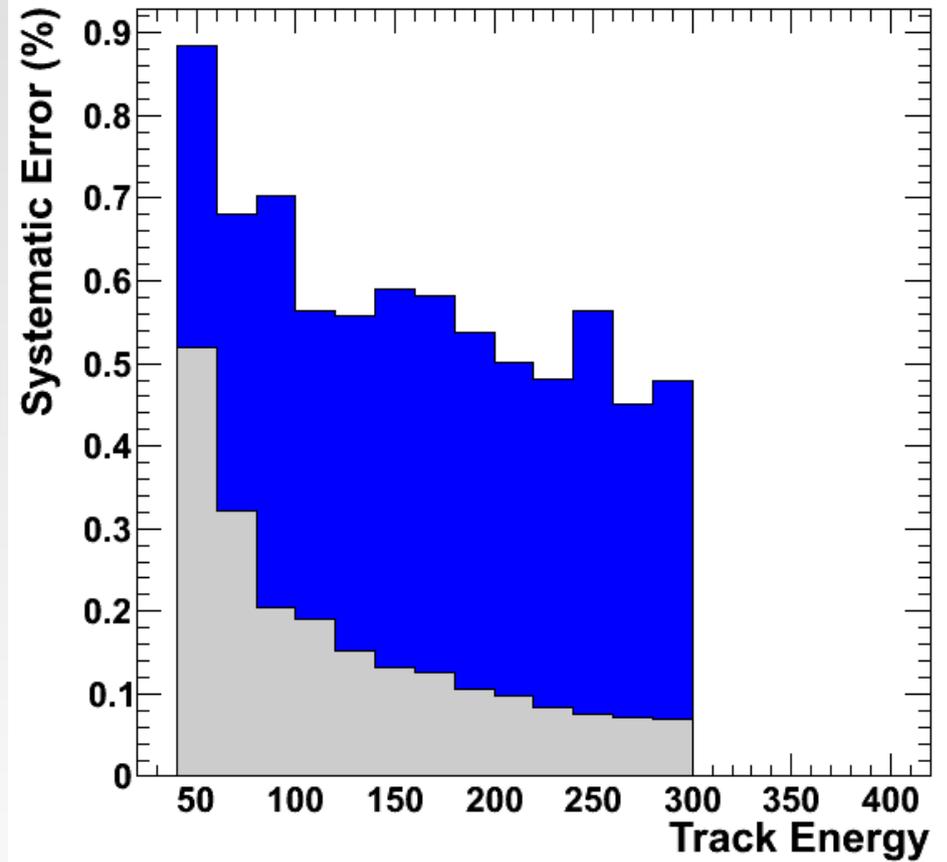
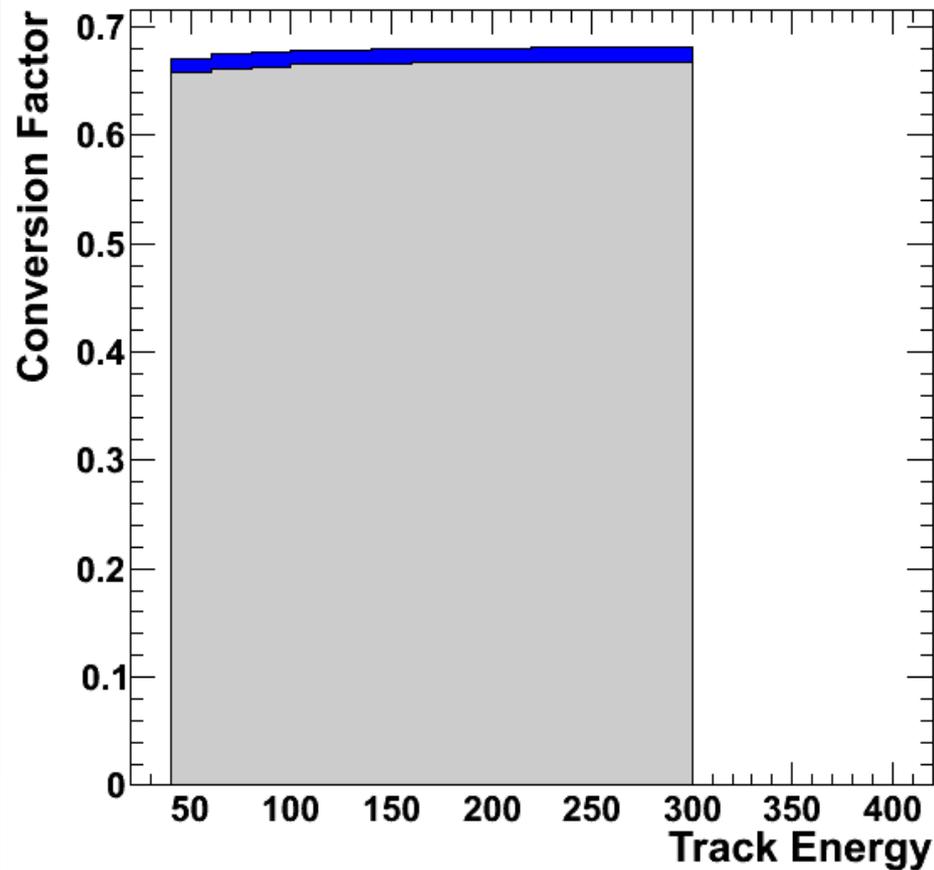


Track Energy



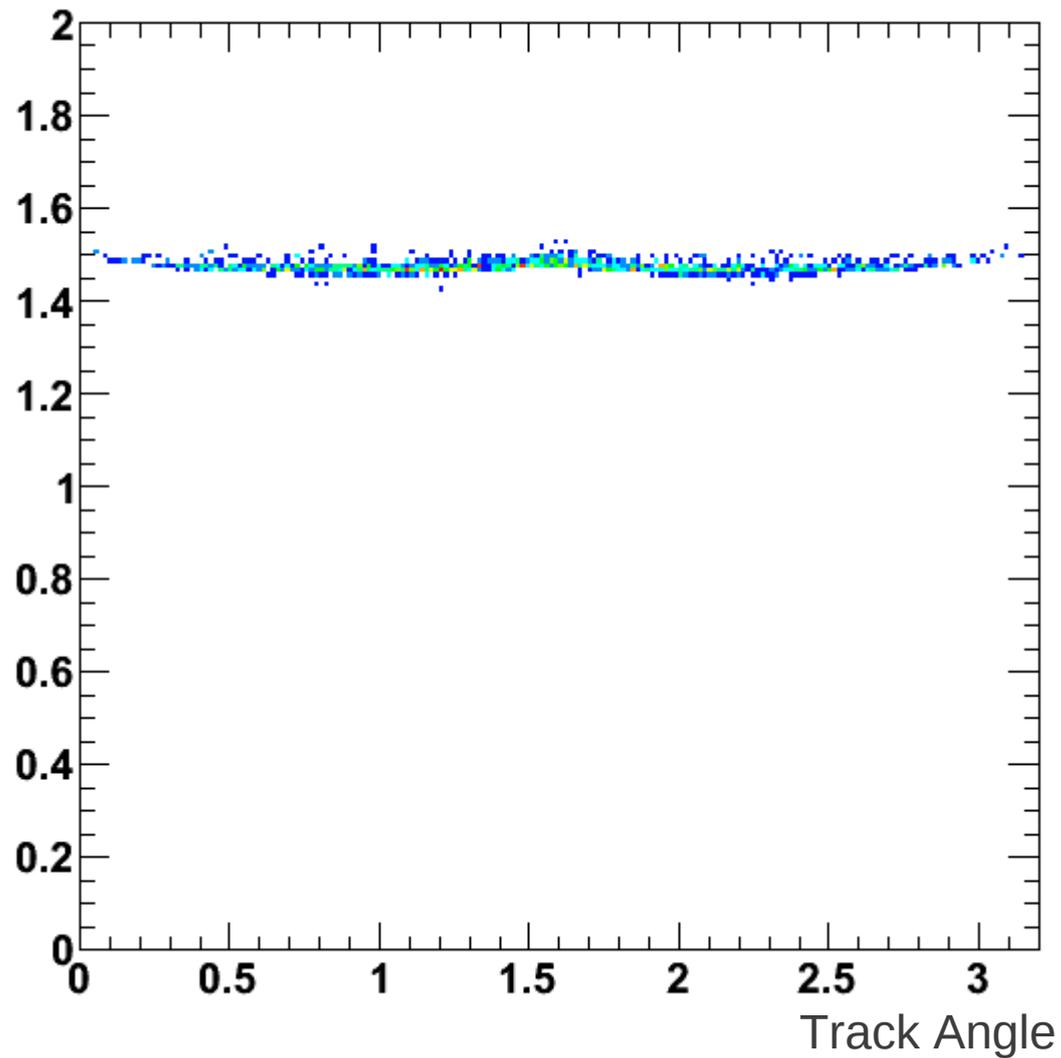
Track Energy

$dE/dx = 2 \text{ MeV / cm}$



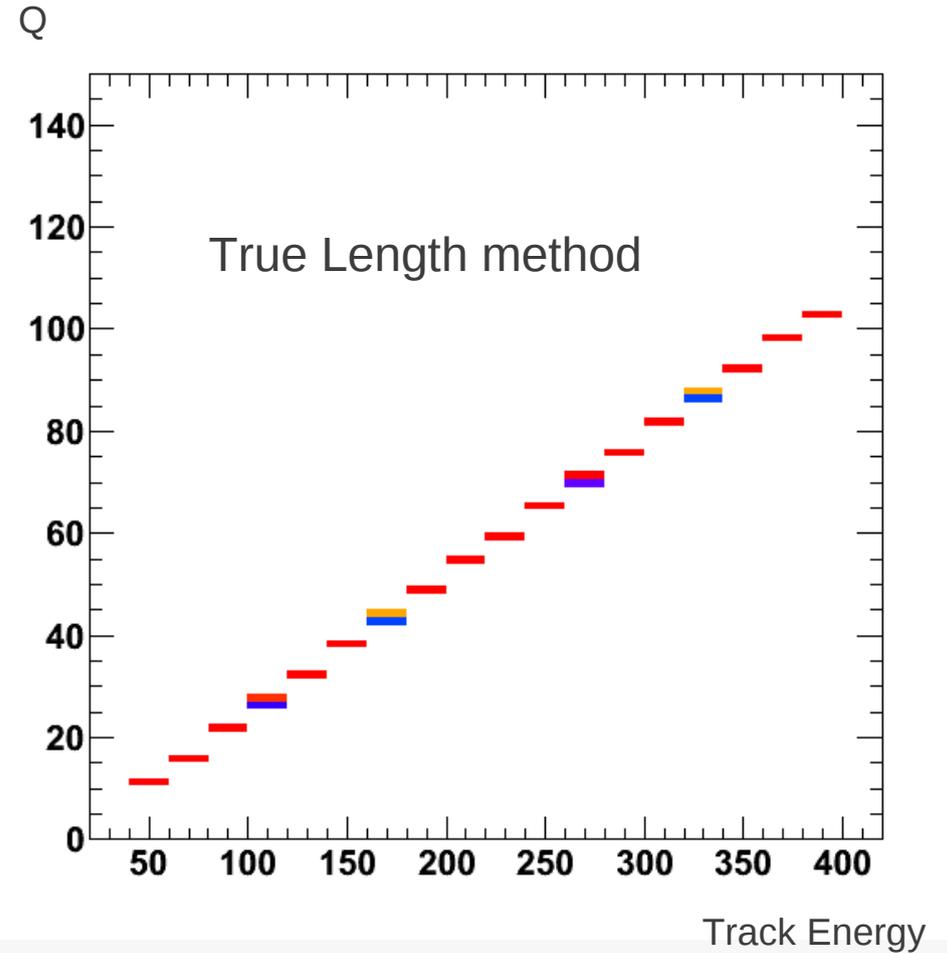
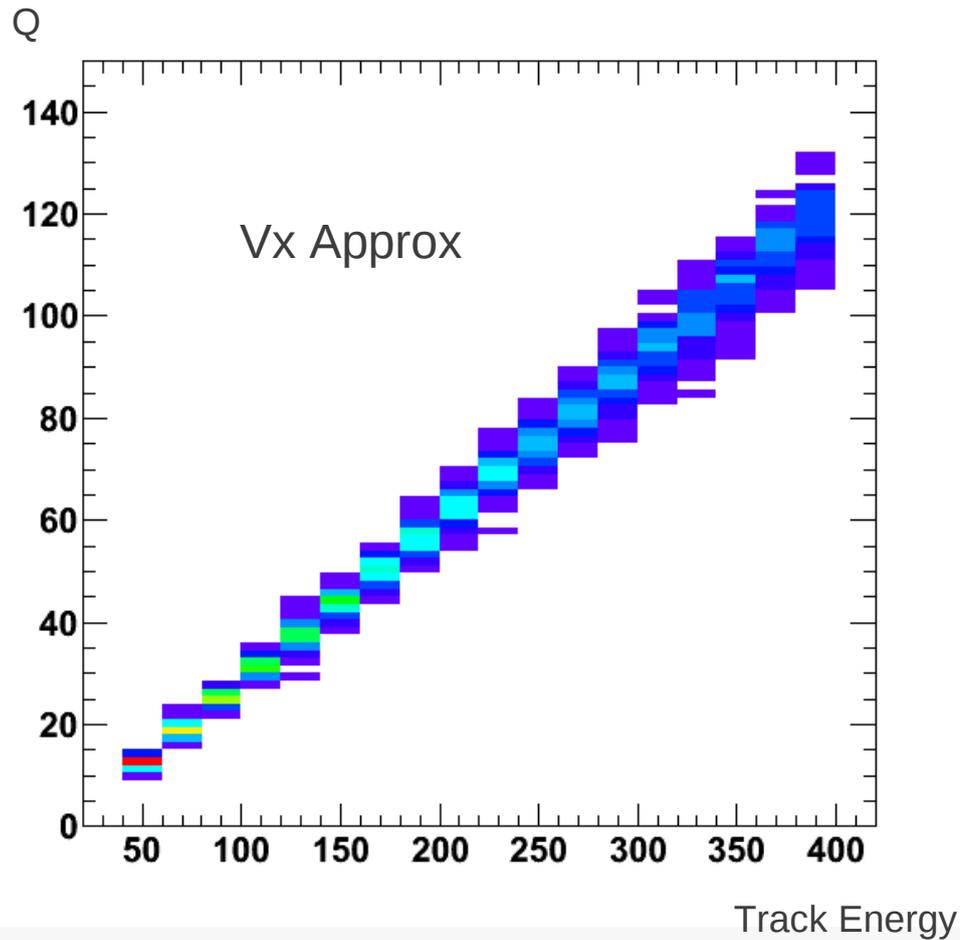
$$dE/dx = 2 \text{ MeV} / \text{cm}$$

Energy / Charge

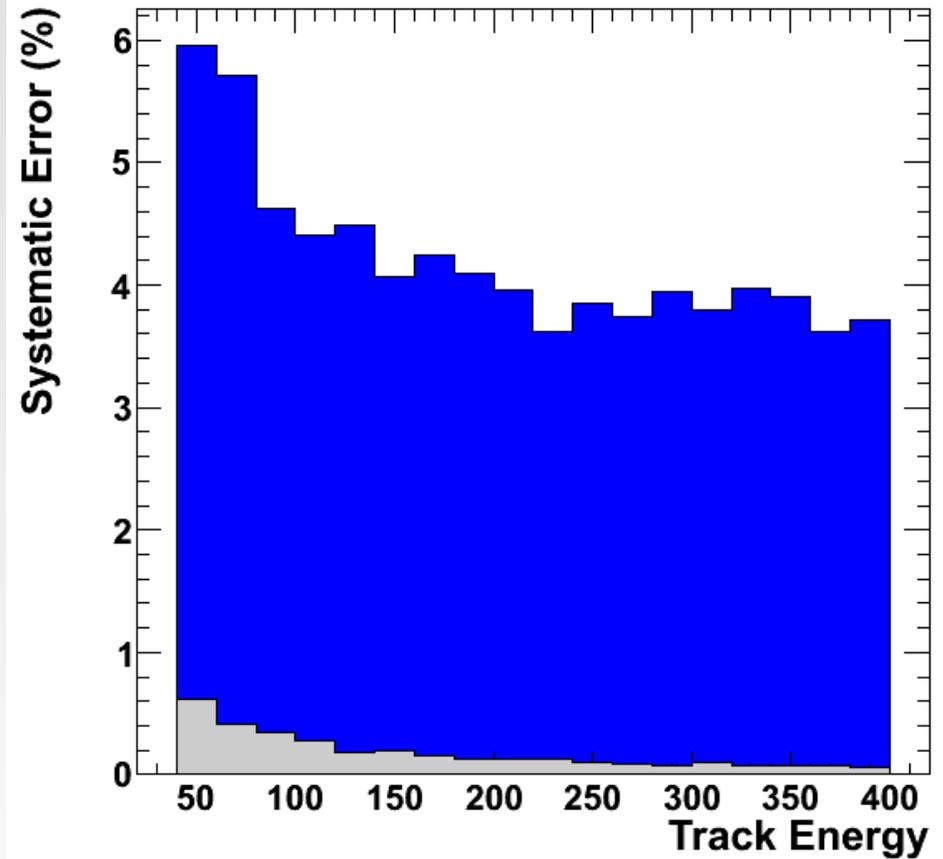
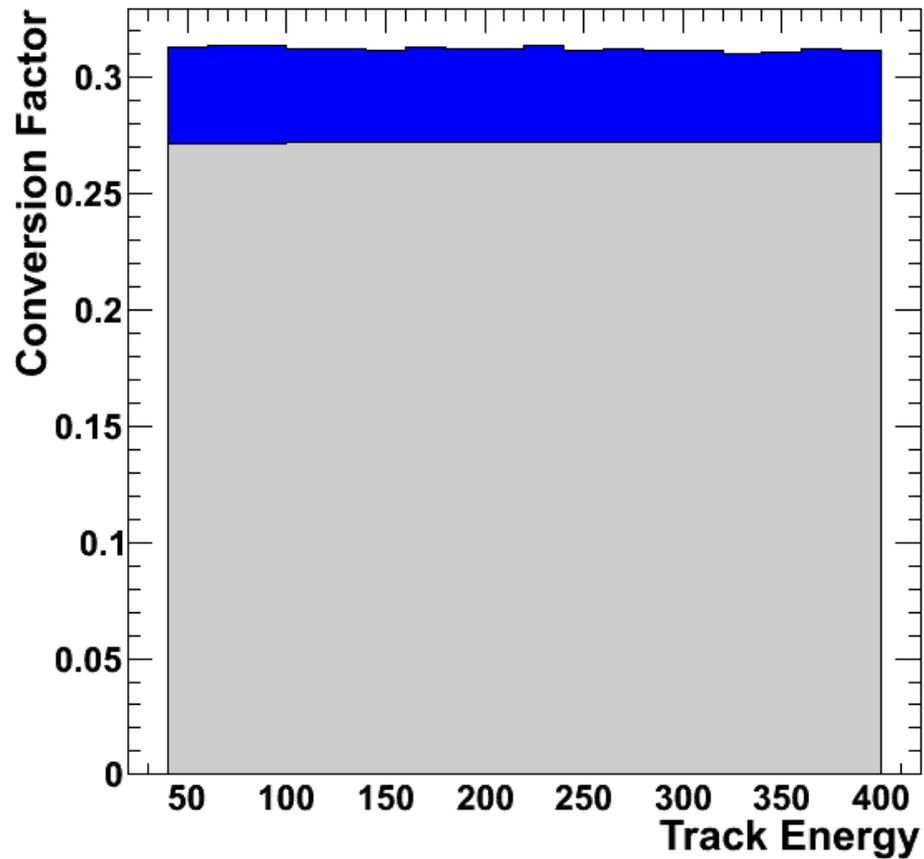


$dE/dx = 20 \text{ MeV / cm}$

$dE/dx = 20 \text{ MeV / cm}$

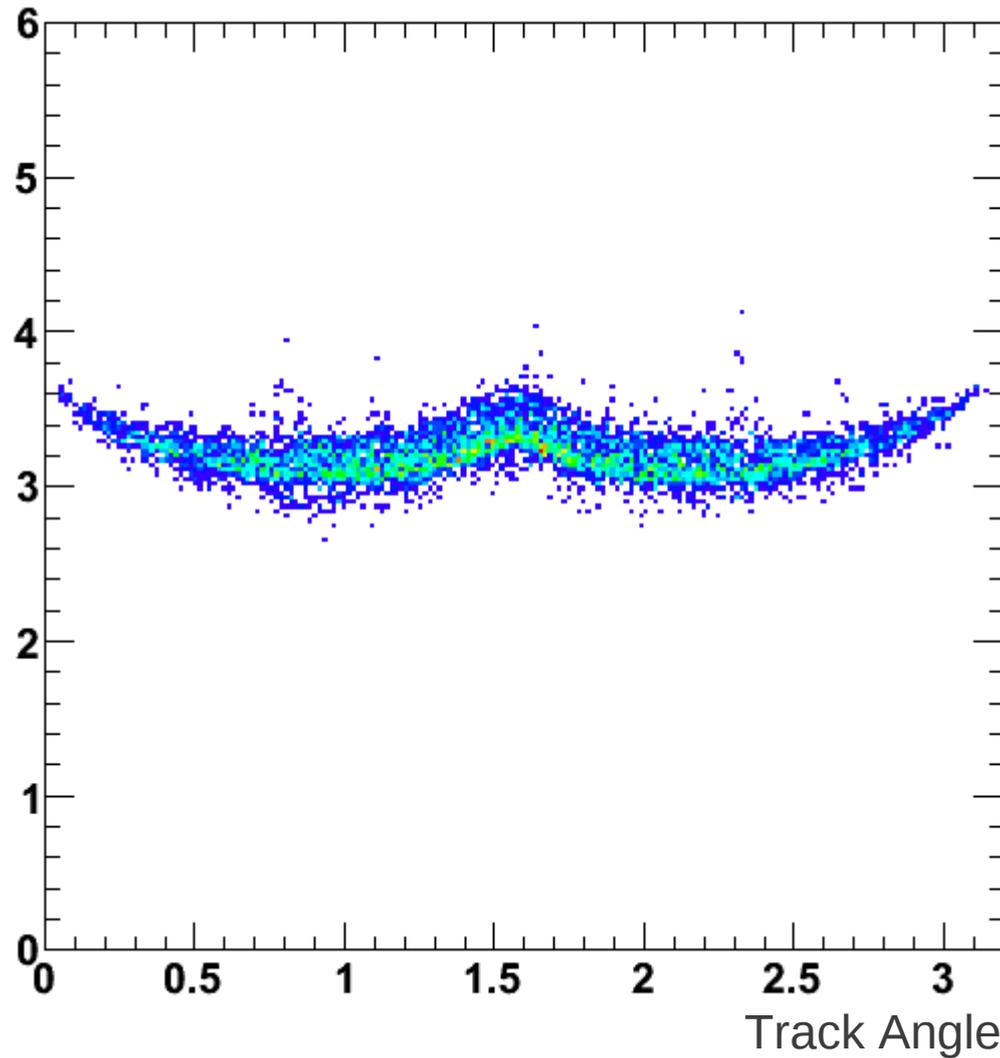


$dE/dx = 20 \text{ MeV / cm}$



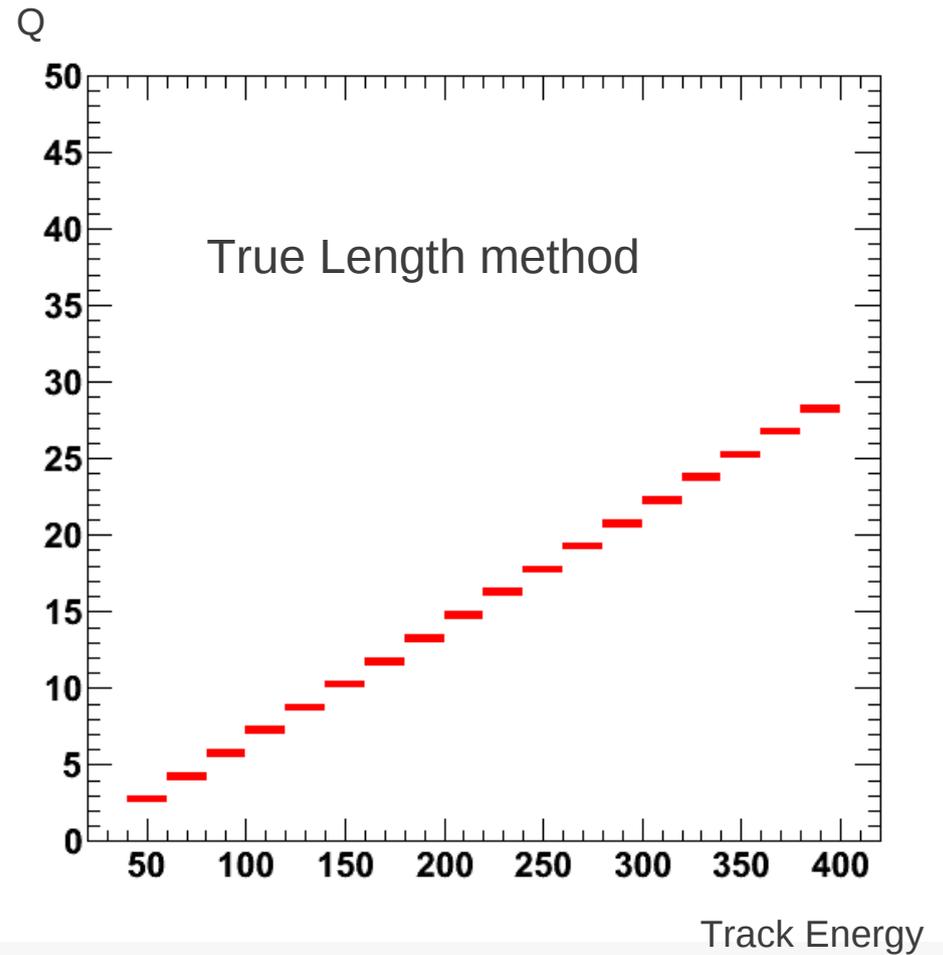
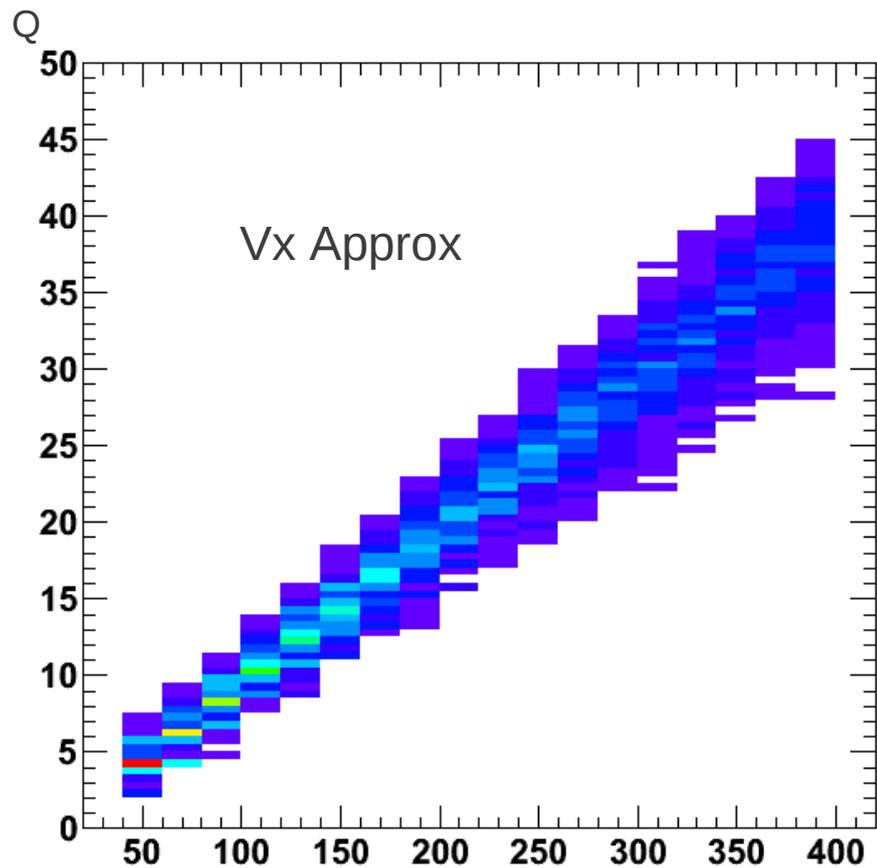
$$dE/dx = 20 \text{ MeV / cm}$$

Energy / Charge

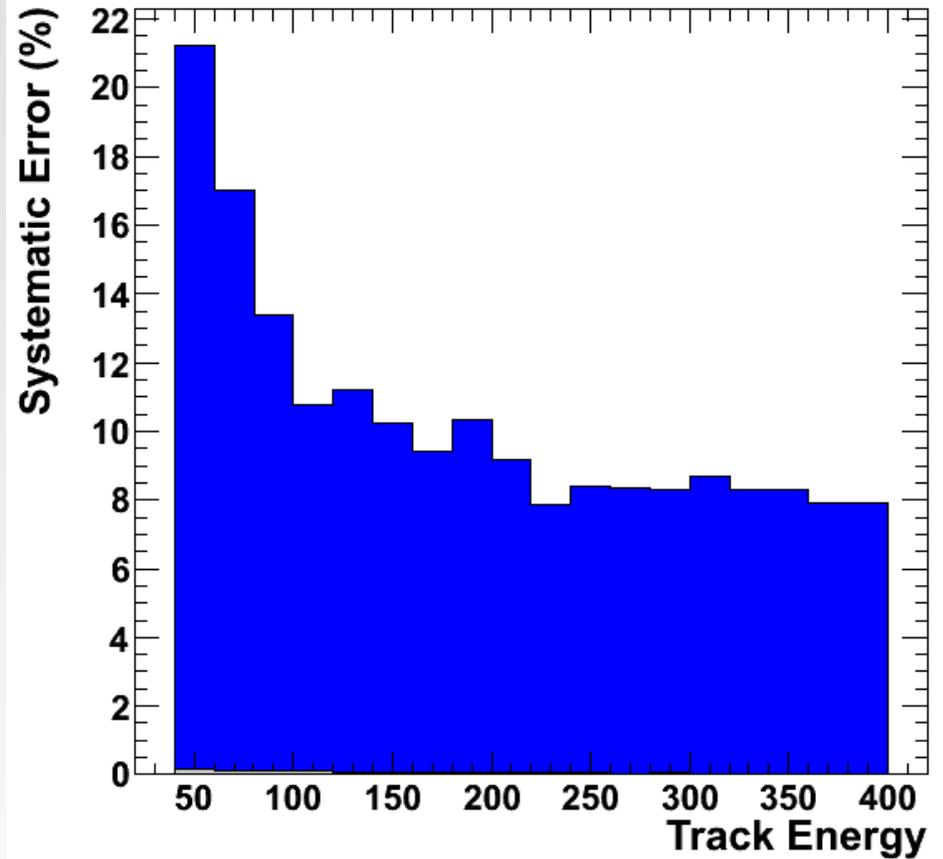
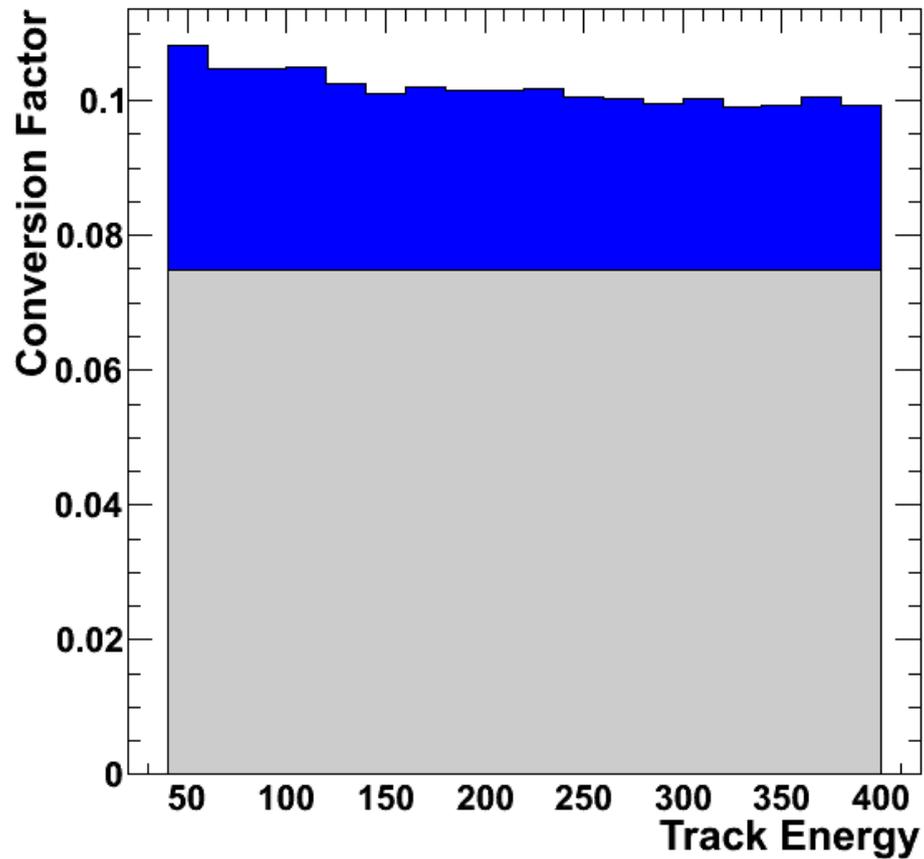


$dE/dx = 100 \text{ MeV / cm}$

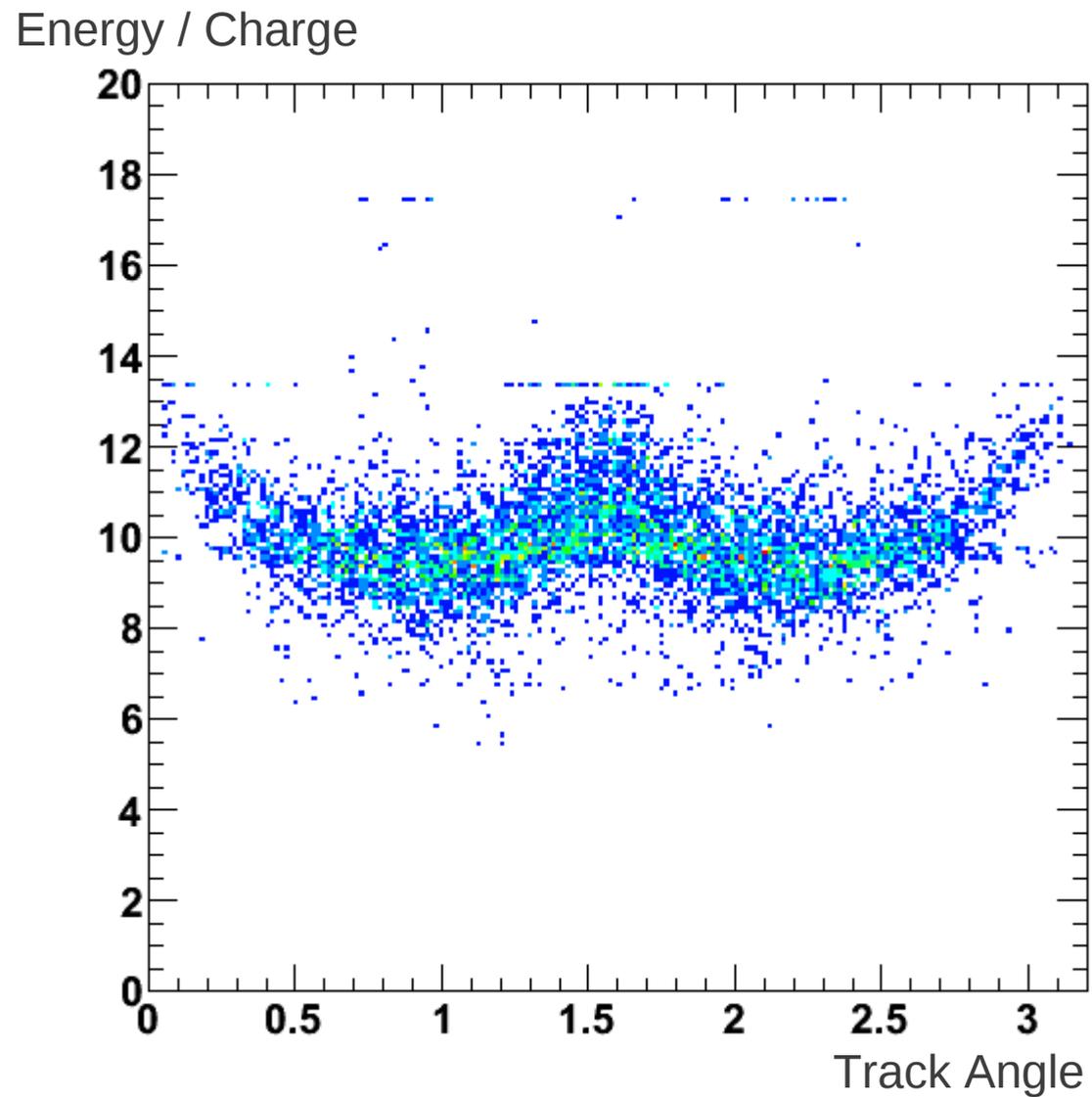
$dE/dx = 100 \text{ MeV / cm}$



$dE/dx = 100 \text{ MeV} / \text{cm}$



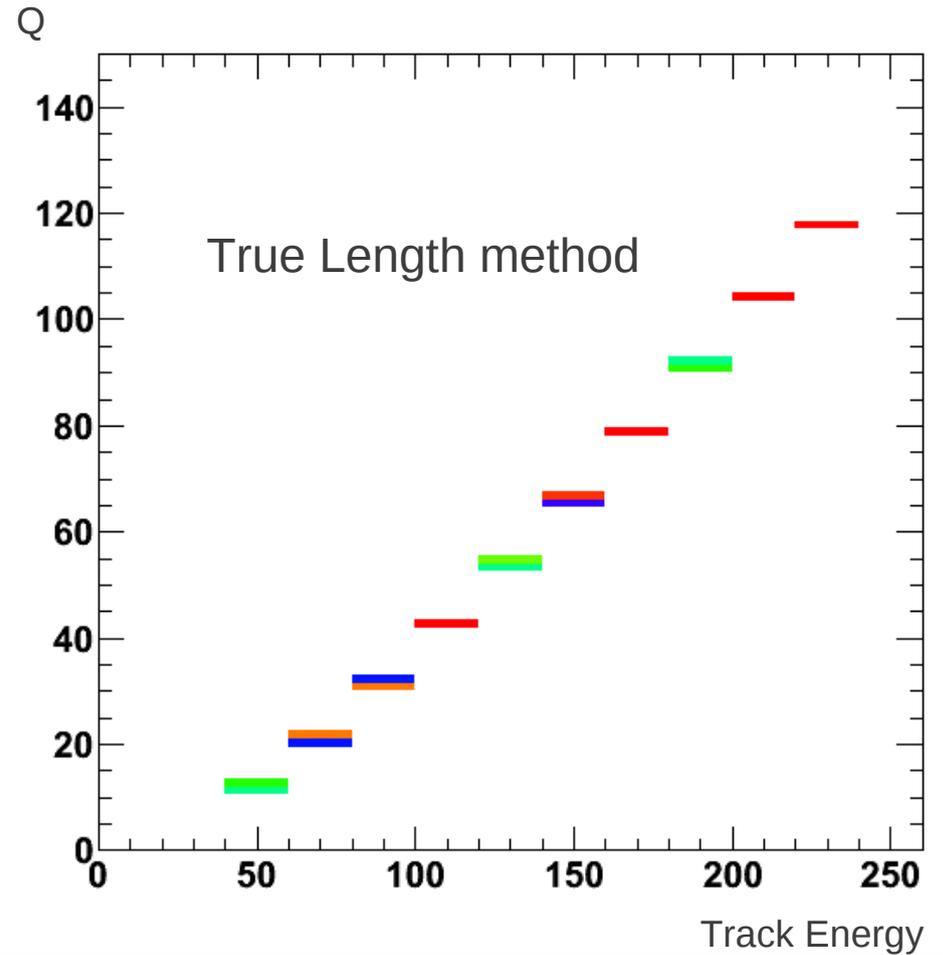
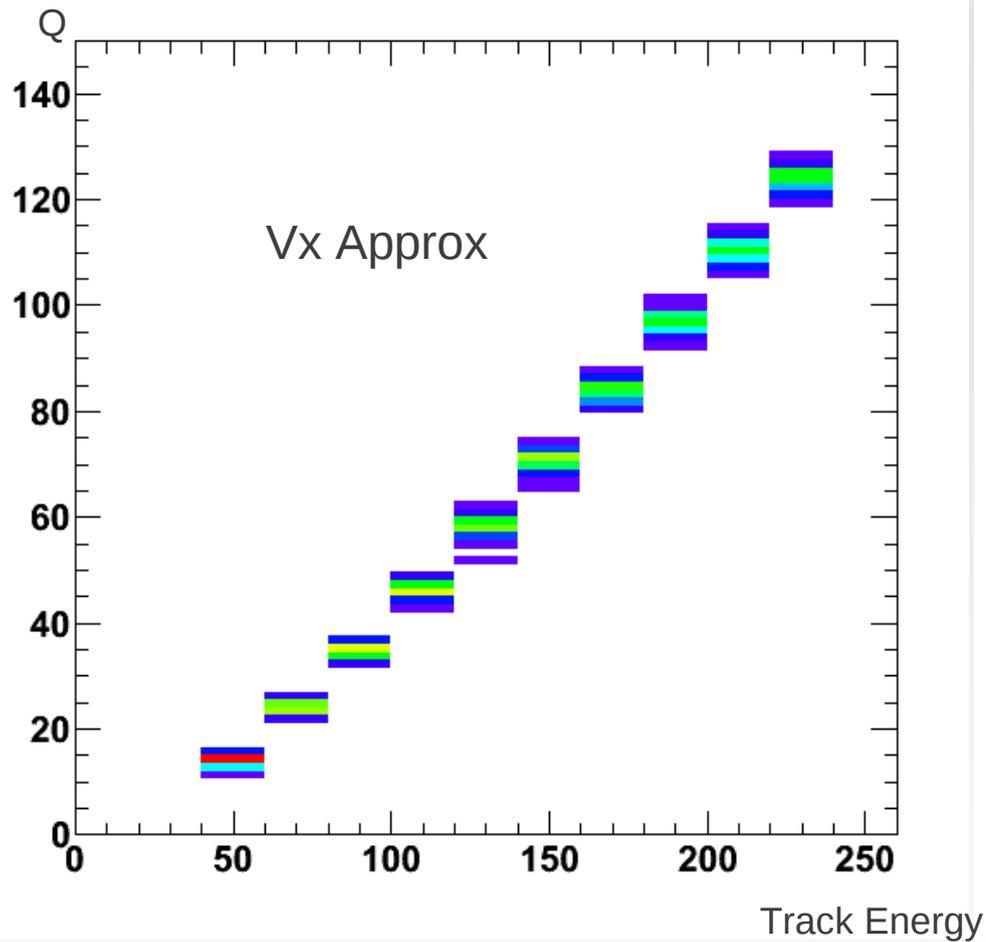
$$dE/dx = 100 \text{ MeV} / \text{cm}$$



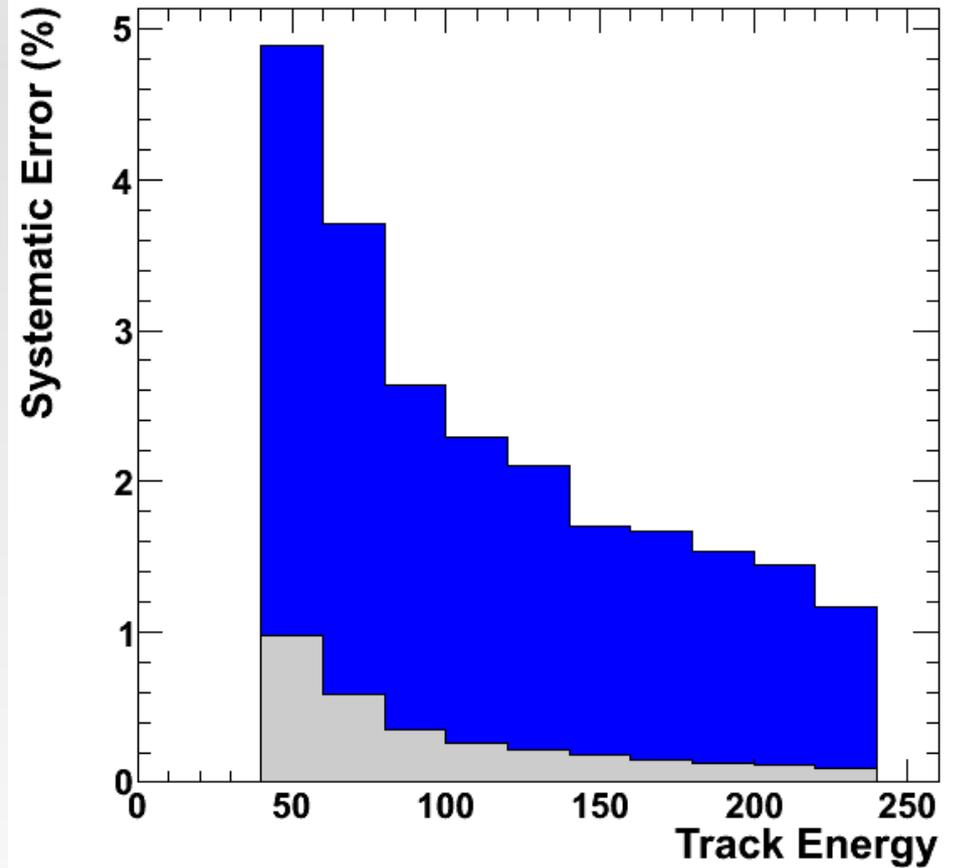
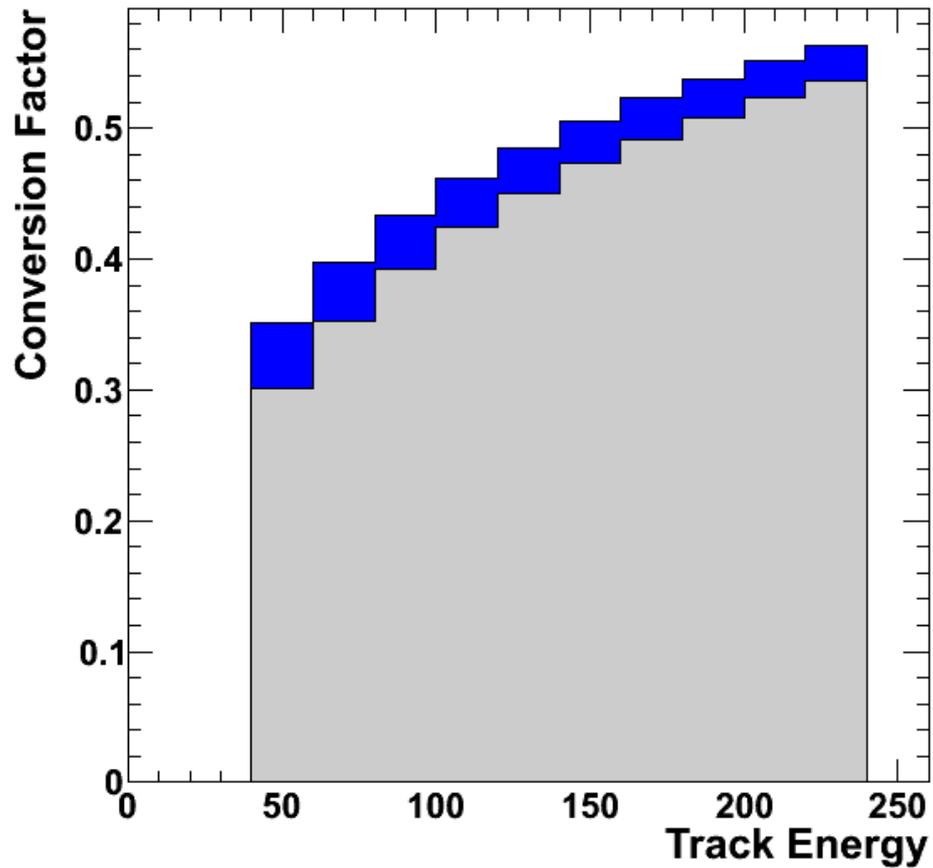
Bethe Bloch Tracks

- The previous results were nice for illustration, but real particle tracks do not have constant dE/dx .
- Instead we apply Bethe Bloch energy losses within the same framework.
- We only bother with highly ionizing particles (here I use protons), since the 2MeV/cm const dE/dx result has already convinced me that the energy measurement for MIPs is not at risk

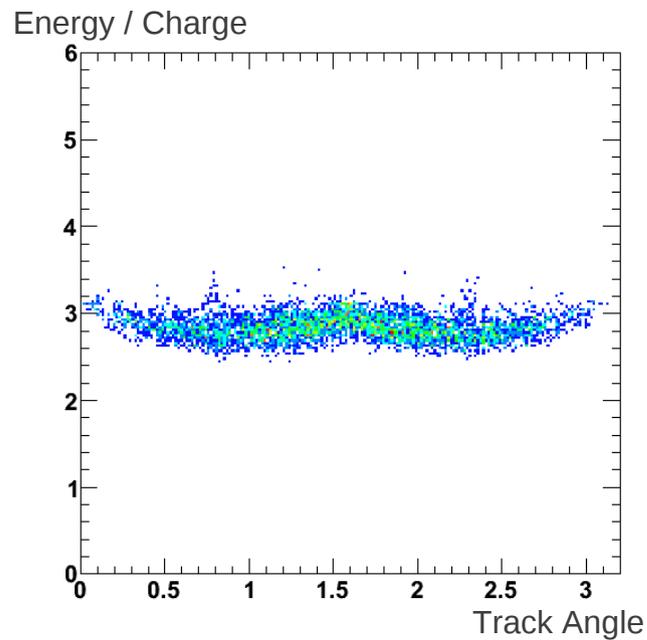
Bethe Bloch Protons



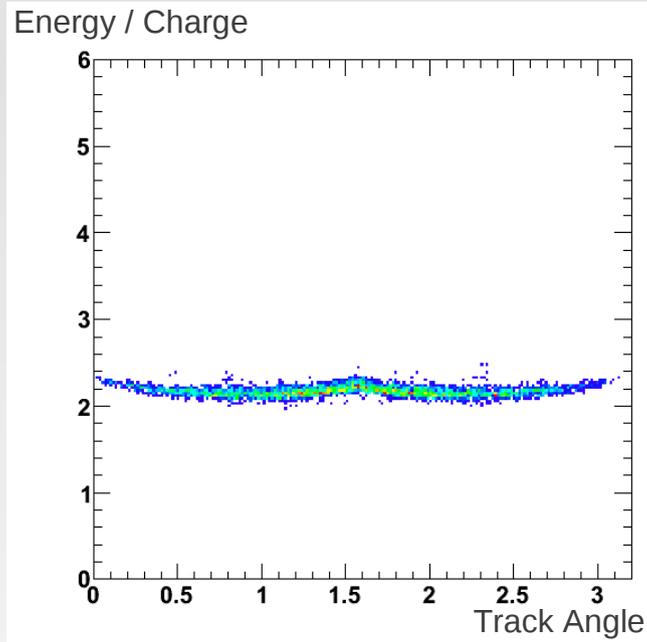
Bethe Bloch Protons



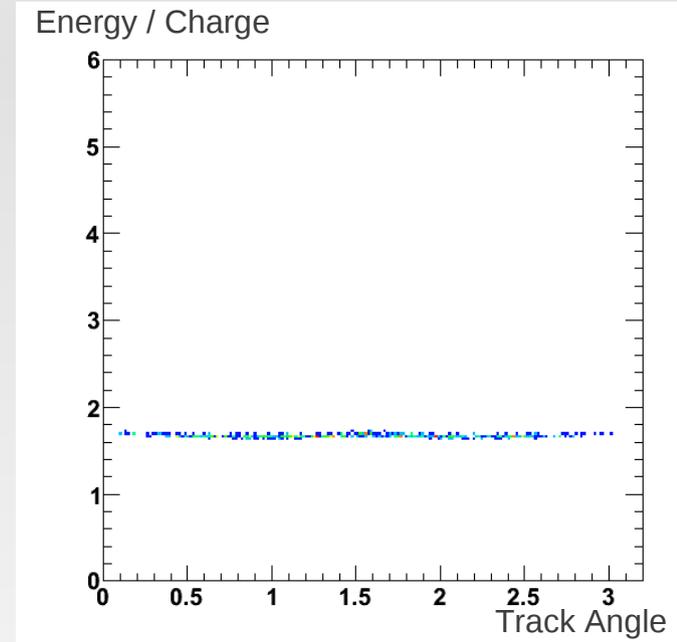
Bethe Bloch Protons



$E = 40\text{MeV}$

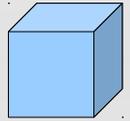
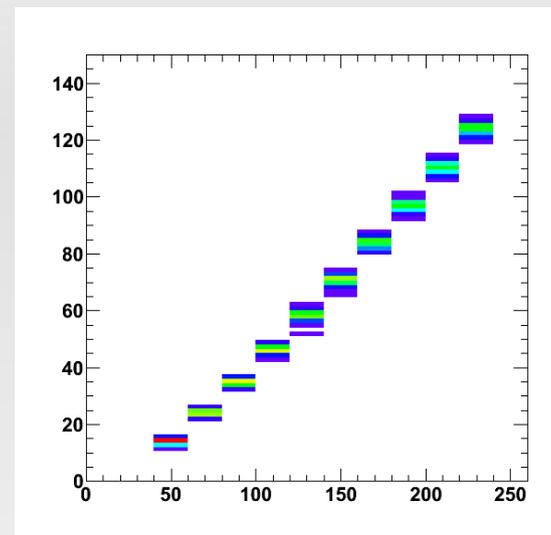
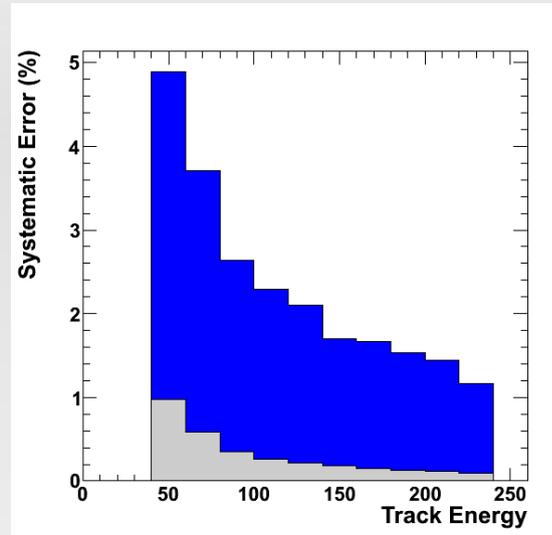
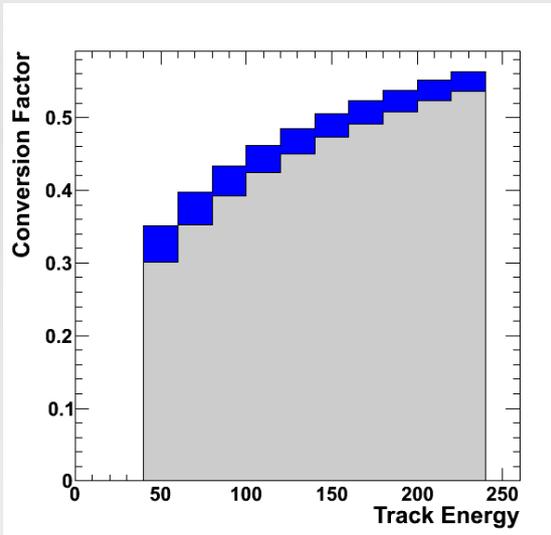


$E = 100\text{MeV}$

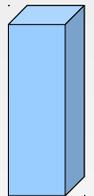
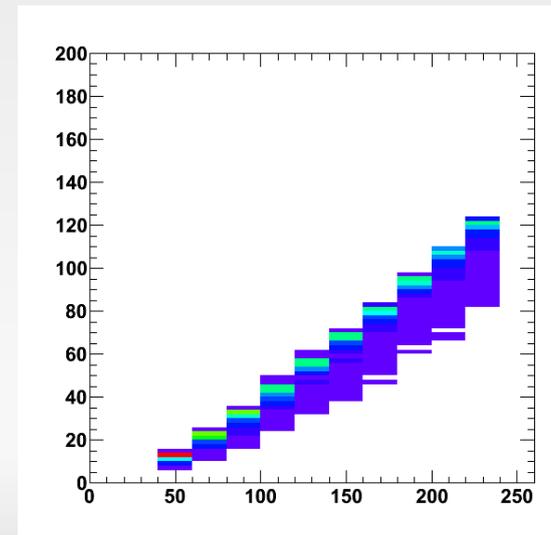
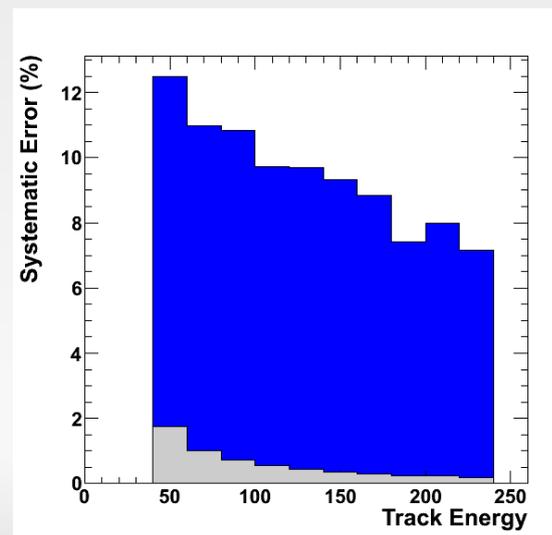
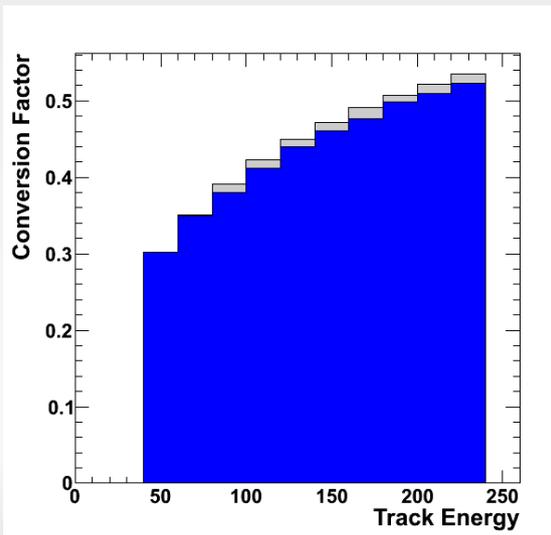


$E = 300\text{MeV}$

Nonisotropic voxels



3 x 3 x 3 mm



2 x 2 x 6.7 mm

Per Voxel Effects

An important note at this point:

- Remember that cumulative charge along the track is not the only important quantity.
- LarSoft reconstruction starts with hitfinding and cluster finding algorithms which have to start out with single wire level information. Hence their efficiencies definitely depend on individual-voxel level effects.
- Obviously the effects will be most visible for low energy tracks where the high dE/dx region is the largest fraction of the track.
- Lets look at the charge-per-voxel distributions for the true length method and the voxel width approximation for different track energies

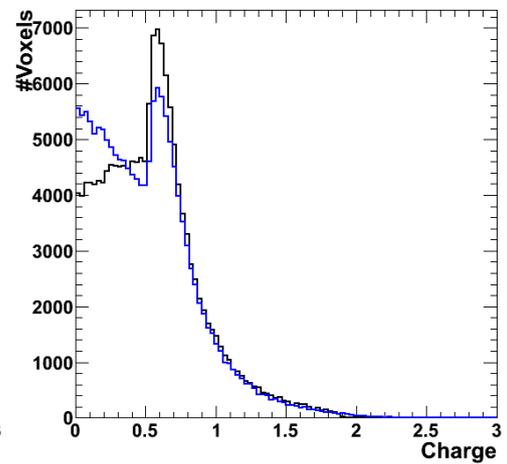
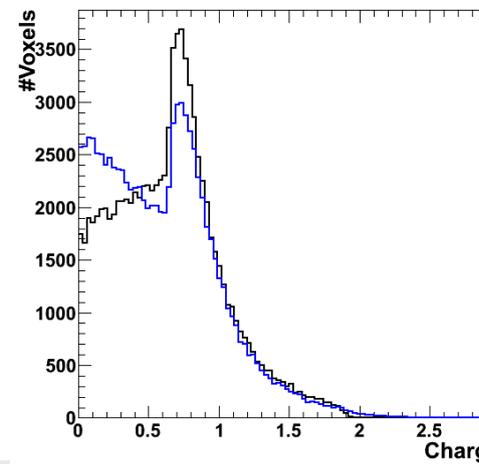
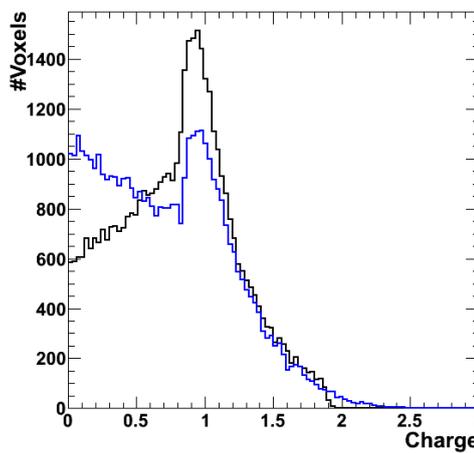
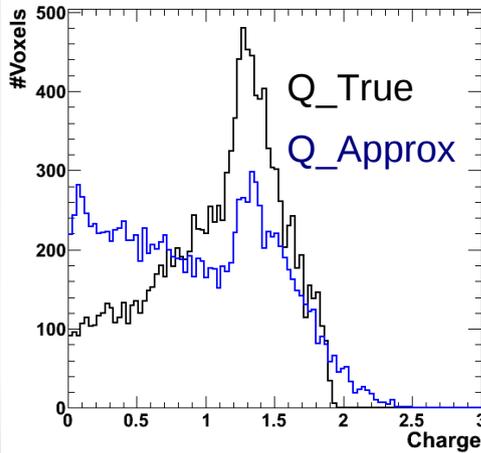
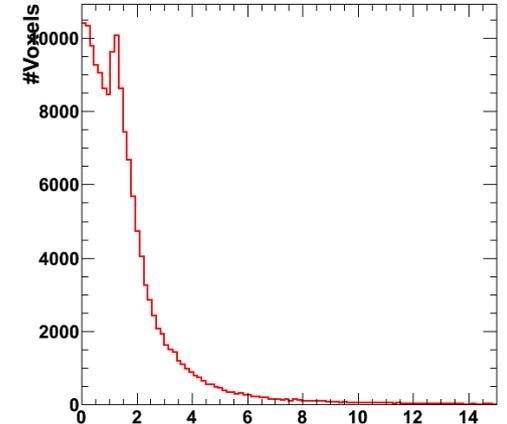
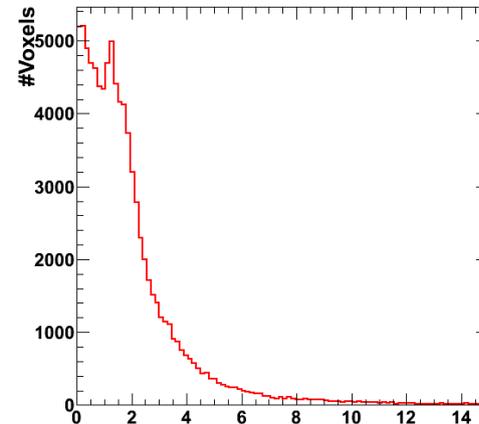
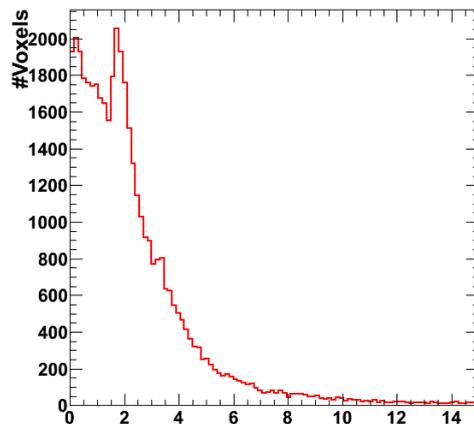
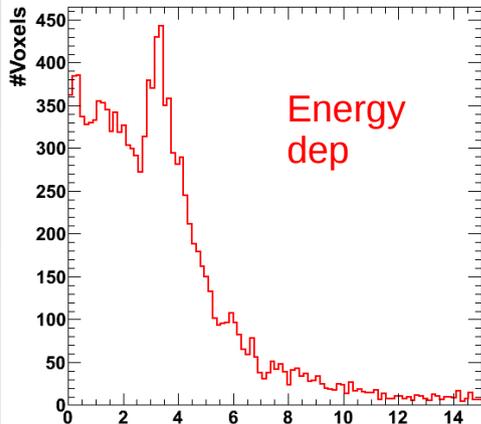
Charge per voxel

40 MeV

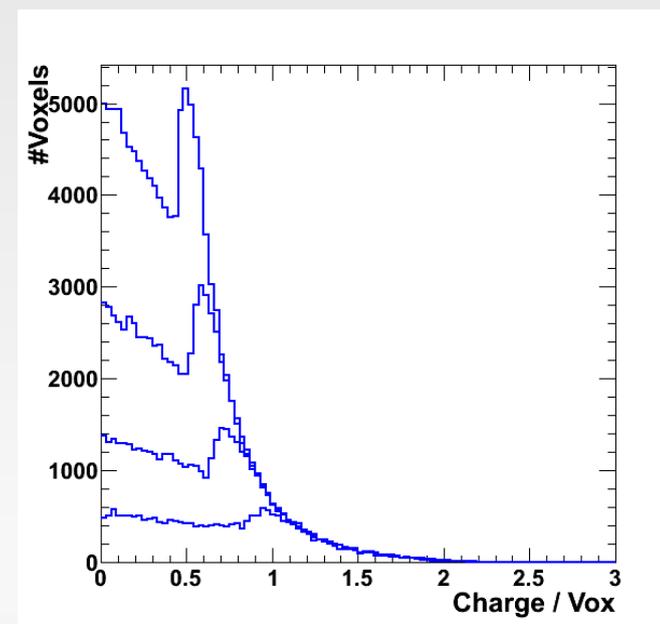
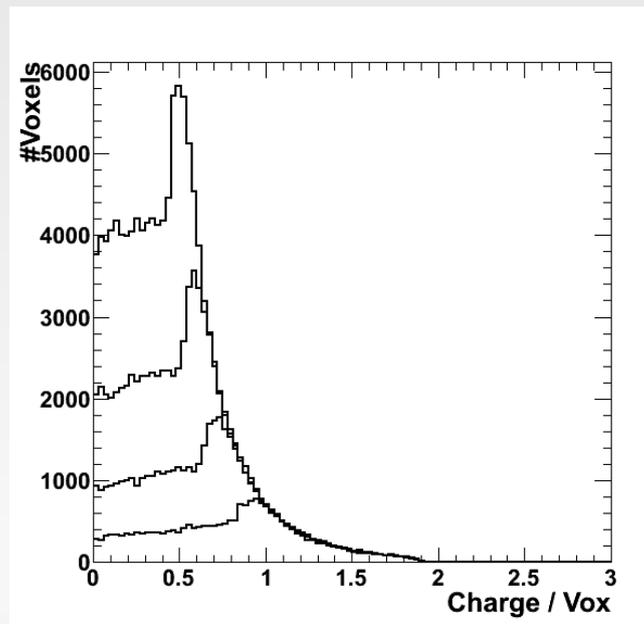
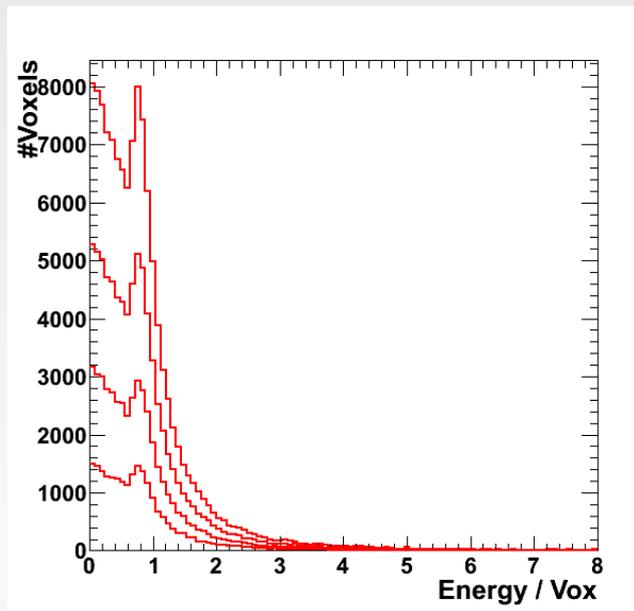
80 MeV

120 MeV

160 MeV

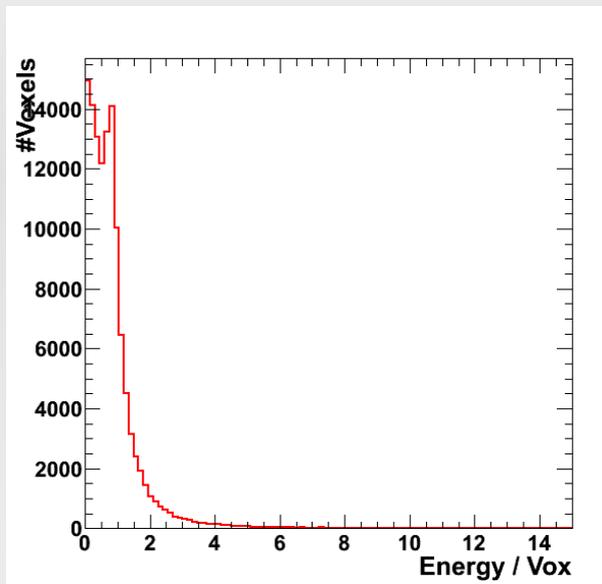


Charge Per Voxel at Different Energies

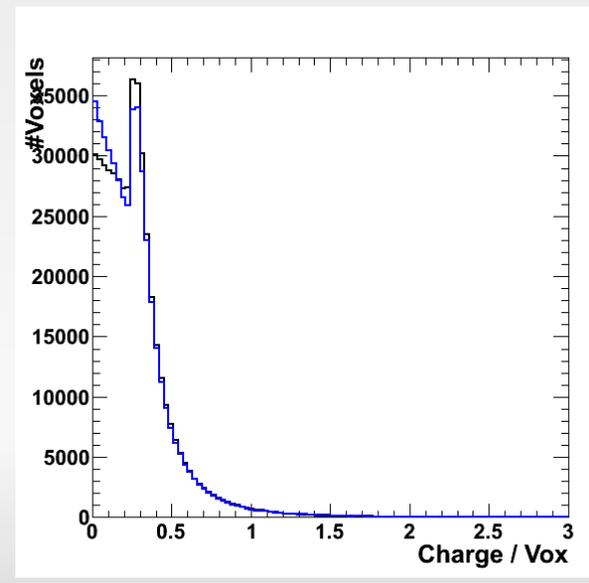
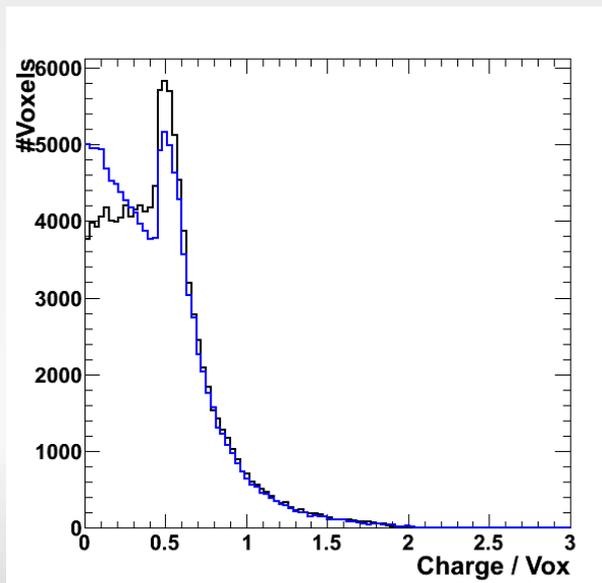
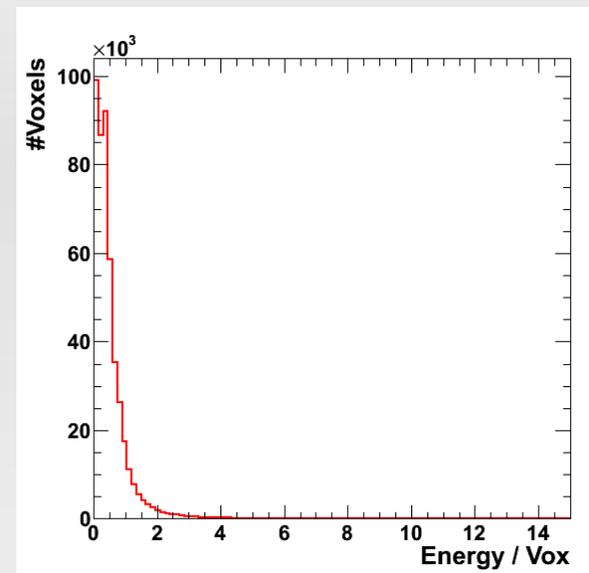


Charge Per Voxel

200 MeV



400 MeV



Summary

The take home message:

Charge measurements of tracks made from summed voxel deposits can be compared between data and monte carlo by folding in a relevant systematic uncertainty of a few % in energy resolution.

BUT

The charge distribution across different voxels is also affected when using this approximation.

DBSCAN and other low level reco algorithms will in general behave differently for data and for voxel-length recombination approximated monte carlo.

Effects on clustering etc may be nontrivial

Efficiencies etc derived from monte carlo cannot be applied to data for non-MIPs, without a detailed study of these sort of effects.