

The DeepCoreVeto Module

Lena Bradley¹, Darren Grant^{2,3}, Olaf Schulz³,
Doug Cowen¹ and Tyce DeYoung¹

¹*The Pennsylvania State University*

²*University of Alberta*

³*Max Planck Institute, Heidelberg*

May 12, 2010

1 Introduction

1.1 DeepCore

Motivation In order to extend the lowest energy threshold of the IceCube detector below about 100 GeV, new strings of high quantum efficiency DOMs have been added to the nominal IceCube-80 (IC80) array. These new strings were deployed in the bottom-center of the detector volume; a region referred to as DeepCore. The DeepCore region is more densely instrumented, increasing the probability that lower energy events will satisfy detector trigger conditions. This new spatial construction also allows the IC80 strings and DOMs to act as an active veto volume for DeepCore. In doing so, the detector can be used to focus on topics such as dark matter searches, neutrino oscillations, and southern hemisphere point source searches.

Geometry The new strings being deployed are numbered 81 through 86 in the base geometry text file used as input to the IceSim Monte Carlo package. The DOMs are divided, as indicated as indicated in Table 1 below, to make up the two separate regions. There is no overlap or sharing of DOMs between IceCube (the veto region) and DeepCore in this definition; the sets are mutually exclusive.

String Numbers	DOM Numbers	
	IceCube, Veto Region	DeepCore
81-86	1-10	11-60
26-27, 35-37, 45-46	1-37	38-60
1-25, 28-34, 38-44, 47-78	1-60	

Table 1: Allocation of DOMs into IceCube and DeepCore fiducial Volumes

DeepCore is centered around current string number 36 which is located at (x, y) coordinates (45.98 m, -34.539 m) with a radius of about 125 m. Vertically, DeepCore will cover the range $-503 \text{ m} \leq z \leq 100 \text{ m}$. Above the dust layer we have a DOM spacing of 10 m instead of 7 m, with the gap at the dust layer which is not instrumented at all. The additions to the detector will be deployed as pictured in Fig. 1. A side view of the final geometry is shown in Fig. 2. A simpler, but less accurate representation can be seen in Fig. 3, which also illustrates how signal and background events are defined for DeepCore: events whose interaction vertices (starting points) are in the defined DeepCore region or below the entirety of IceCube are considered signal for that fiducial volume, while all other events are considered background.

1.2 Background Concerns

Atmospheric muons represent the dominant background for low energy analyses in IceCube. Estimates from CORSIKA simulation have shown an expected rate of atmospheric muons of 10^7 per hour. This is compared to an expected signal rate for muon neutrino events (triggering only the fiducial volume) of approximately 10 per hour. Thus, the ability to achieve a rejection level of 10^7 or better against atmospheric muons would yield a signal-to-noise ratio of better than 10:1, making analyses at the lowest energies possible.

1.3 Veto Algorithm Goals

In designing such a rejection (or veto) algorithm to run online, the goal is to achieve background elimination to satisfy satellite bandwidth constraints while maintaining the highest possible signal efficiency. This will be done by discarding those events that do not appear to have interaction vertices within the DeepCore volume and keeping those that do. A computationally

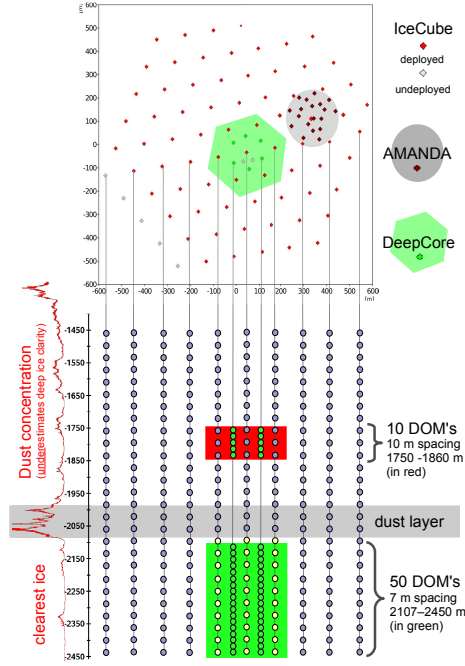


Figure 1: A schematic layout of IceCube DeepCore. The upper diagram shows a top view of the string positions in relation to existing AMANDA and current/future IceCube strings. It includes two additional strings, situated close to the central DeepCore string, that are planned for deployment in the 2010/2011 austral summer. The lower diagram shows the instrumented DeepCore region (highlighted in red and green) with the surrounding IceCube strings. On the left is a dust concentration curve which is related to the effective scattering length for Cherenkov photons in the ice. The curve is from data down to 2100 m depth, and an extrapolation thereafter and then extrapolation thereafter because the ice below 2100 m becomes too clear for the device that made the measurements.

innocuous algorithm of this type will be applied at the pole as a level zero filter. Rejection of the final order of magnitude or so of background will naturally come at some expense in signal efficiency and will be accomplished using sophisticated, time-consuming algorithms run offline in the north.

2 The Algorithm

2.1 Overview

The veto algorithm is applied through the use of timing, charge, and position information for all the Hard Local Coincidence (HLC) DOM hits in an event. If an event causes hits in both DeepCore and IceCube (i.e., in the veto region), then veto region hits are compared to those in DeepCore, searching for a correlation. Location and timing information from the hits are used to search for a coincidence within a certain time window which is centered around light travel time $t = d/c$ between hits in the veto region and those in the fiducial region (d is defined below). If such a correlation is found then the event in question is discarded. By default, events that do not cause hits in DeepCore DOMs at all are discarded while those that produce hits in only DeepCore DOMs are kept.

2.2 General Process

The module first checks if the event produced any hits in DeepCore, a process which will be executed by the DeepCore trigger module. If an event does not cause DeepCore hits, then it is discarded. Otherwise, the algorithm focuses first on the DeepCore fiducial hits, and calculates the mean and standard deviation of the timing distribution for all DeepCore hits (See Appendix B.1). From this distribution, hits that fall within one standard deviation of the mean are used to determine the DeepCore center of gravity (COG) for the event. The timing of each of these DOMs is used to calculate a time for the COG, corrected for the COG-DOM distance and the speed of light in the ice. Appendix B.2 details both of these calculations and the specific formulae are repeated below in equations 1 and 2 using the option of weighting the hits by charge. The filter that is run at the Pole does **NOT** weight the hits by charge, but the mechanics to do so are included in the filter module. For completeness the equations shown here include charge weighted hits.

For a set of DeepCore hits with position vectors $\langle x_i, y_i, z_i \rangle$ and charge

deposited q_i , the COG position vector has the components

$$X = \frac{\sum_i^n (q_i x_i)}{\sum_i^n (q_i)}, \quad Y = \frac{\sum_i^n (q_i y_i)}{\sum_i^n (q_i)}, \quad Z = \frac{\sum_i^n (q_i z_i)}{\sum_i^n (q_i)} \quad (1)$$

If each DeepCore hit also has a hit time given by t_i and the speed of light in ice is given by c_{ice} then the weighted time for the COG is

$$\text{Weighted COG Time} = T_{cog} \frac{\sum_i^n |(t_i - \frac{\sqrt{(x_i - X)^2 + (y_i - Y)^2 + (z_i - Z)^2}}{c_{ice}})|}{\sum_i^n t_i} \quad (2)$$

Then each individual veto region hit is examined and a representative velocity (or “particle” speed) is calculated. This is the velocity that an imaginary particle would need to have in order to satisfy the difference between position and timing between the specific veto region hit and the calculated COG as shown in Eq. 3 below.

$$\text{Particle speed} = \frac{\sqrt{(x_i - X)^2 + (y_i - Y)^2 + (z_i - Z)^2}}{T_{cog} - t_i} \quad (3)$$

Within the particle speed distribution a pre-determined “veto window” is defined. This window, which is used as a cut on the data, is based loosely on the speed of light in vacuum that an energetic muon would possess in the detector, such as those from atmospheric backgrounds. (This window used was not computed through any formulae but rather determined via trial-and-error.) If greater than n veto region hits produce a particle speed that falls within this window then the entire event is vetoed (See Appendix B.4). Note that n should generally be very low for contained signal events, $n \leq 1$. If there are not enough coincident hits found after analyzing all veto region hits, then the event is not vetoed.

In the end, information about the veto, center of gravity, and other calculated values may be output by the module to a ROOT file before moving to the next event, if the user specifies to do so in the parameters.

2.3 Further Notes

The particle speed is constructed so that hits occurring before the DeepCore center of gravity hit will have positive speed, while those with a later timing will have negative speeds. In this way, the window can differentiate between those two situations. Also note that even though the particles obviously do not travel faster than the speed of light in vacuum (0.3 m/ns) the windows used in the veto go beyond that range. This is done to allow for PMT jitter or low resolution in the DeepCore COG calculation.

Additionally, this module was built to operate within the IceRec V02-01-01 release and code reviewed and included in IceRec V03-02-00. Source code hierarchy, configurable parameters and other methods of controlling the algorithm are detailed in Appendix A.1, A.2, and A.3, respectively.

3 Results

3.1 Veto Power

Table 2 below details a selection of statistical results obtained by running various test files through the veto, using Hard Local Coincidence (HLC). Note that for corsika data, the result given is in:

$$\frac{\text{number of events kept}}{\text{total number of events}}$$

while for the signal data, results are given in:

$$\frac{\text{number of events kept}}{\text{total number of events with interaction vertexes in DC}}$$

3.2 DeepCore Center of Gravity Calculation

The graphs presented in Figs. 4-6 represent the frequency distribution of the shortest distance between the calculated COG and either the interaction vertex or the track of the particle, as determined from the Monte Carlo (MC) truth information.

Window ($\frac{m}{ns}$)	Data	Hits Allowed in Veto Window	
		0	1
0.25-0.40	Corsika	$\frac{460}{964063} = 4.77 \pm 0.22 E^{-4}$	$\frac{632}{382935} = 1.65 \pm 0.066 E^{-3}$
	ν_e Signal	$\frac{785}{861} = 91.2 \pm 3.3 \%$	$\frac{840}{861} = 97.6 \pm 3.4 \%$
	ν_μ Signal	$\frac{7582}{8496} = 89.2 \pm 1.0 \%$	$\frac{8298}{8496} = 97.7 \pm 1.1 \%$
0.25-0.45	Corsika	$\frac{735}{1501545} = 4.89 \pm 0.18 E^{-4}$	$\frac{607}{1271845} = 4.77 \pm 0.19 E^{-4}$
	ν_e Signal	$\frac{772}{861} = 89.7 \pm 3.2 \%$	$\frac{795}{822} = 96.7 \pm 3.4 \%$
	ν_μ Signal	$\frac{6761}{7704} = 87.8 \pm 1.1 \%$	$\frac{6715}{7626} = 88.1 \pm 1.1 \%$

Table 2: Effectiveness of algorithm on the three data types when using different veto windows.

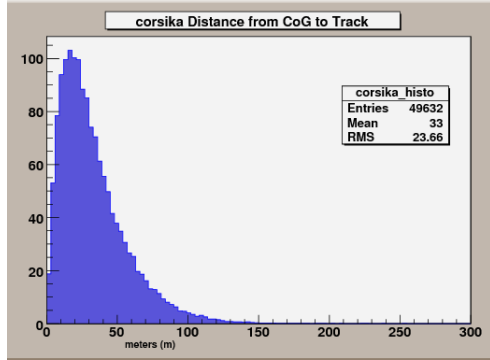


Figure 4: Corsika Distance from CoG to known MC track

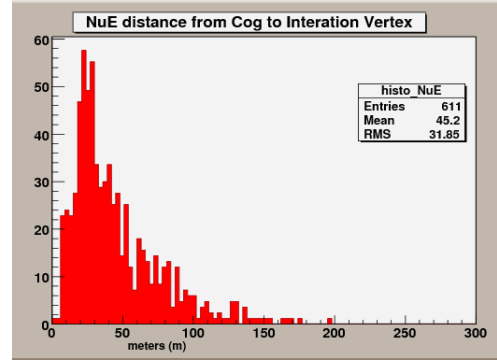


Figure 5: ν_e Distance from CoG to known MC interaction vertex.

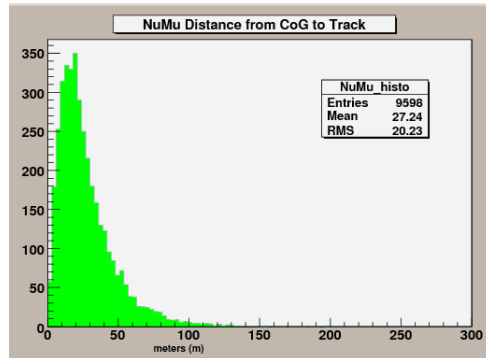
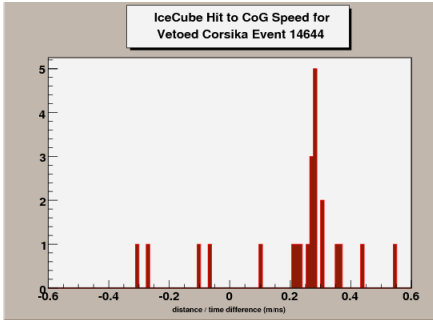
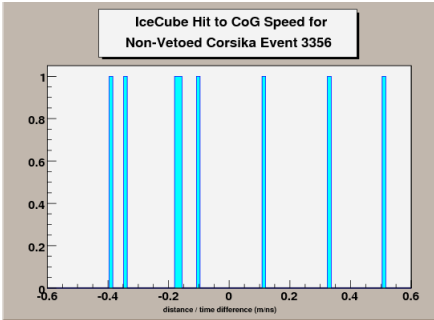
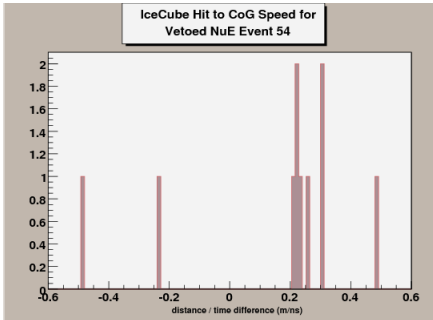
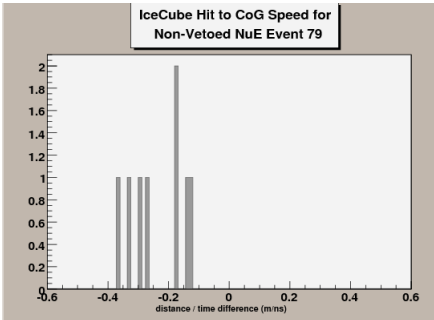
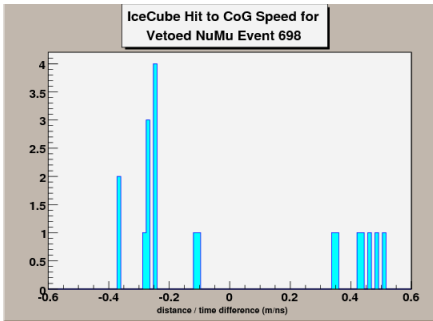
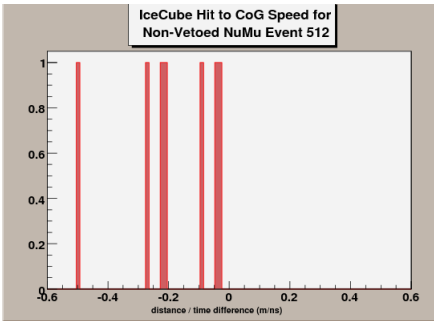


Figure 6: ν_μ Distance from CoG to known MC track.

3.3 Typical Speed Distribution for Veto Region Hit to Center of Gravity Position

The following graphs show the speed calculated by dividing distance between a veto region hit and the CoG by the time difference between those points. For each data type, the particle speed distribution of an example event is given, including vetoed and non-vetoed events. The various possible veto windows (specifically the two possibilities detailed in Table 2) apply to these graphs.

Data	Example Vetoed Event	Example NonVetoed Event
Corsika		
		
NuMu		

4 Conclusions and Future Direction

Used with Hard Local Coincidence, and a 0.25-0.40 m/ns speed window allowing one hit, this veto algorithm can reduce the background by a factor of 1.7×10^3 , while preserving over 97% of the signal. As such, it could effectively

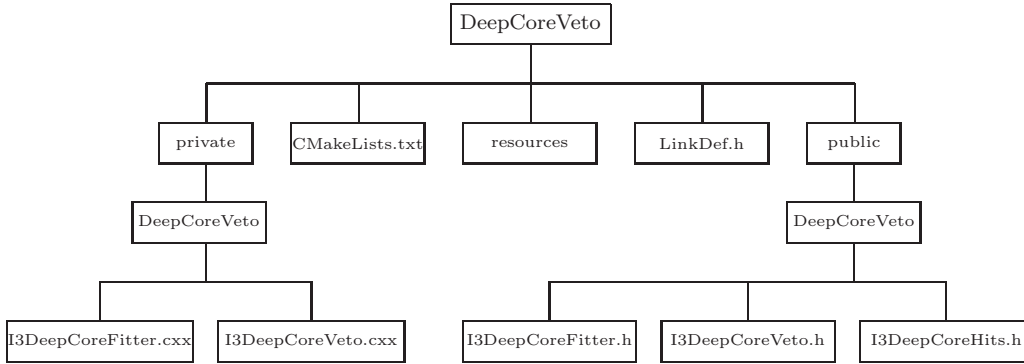
be deployed at the pole as a level-zero pass filter that would consume little bandwidth. Later, higher-level analyses of the data should be able to further reduce the background to attain the final desired 10^{-7} background rejection factor.

The use of HLC with this module does limit the amount of information (hits) available for the veto to analyze, thus limiting its potential as compared to a usage with Soft Local Coincidence. In developing this algorithm, however, working with SLC allowed far too much noise into the data. Basic attempts to reduce noise by increasing n , the number of hits allowed in the veto window, helped somewhat but could not approach the noise reduction necessary for an effective veto and so HLC was used instead.

Although the module utilizes the events after HLC, an application to SLC data has not been refined. The use of SLC has the potential to greatly increase the effectiveness of the algorithm and so there are plans to explore this with topological additions. Further attempts to eliminate the noise inhibiting the exploitation of a SLC analysis could hopefully open new levels of efficiency in both the vetoing power and the signal preservation of this algorithm.

A Technical Implementation

A.1 Source Code Hierarchy



Note: `I3DeepCoreFitter.cxx` contains all of the individual functions, while `I3DeepCoreVeto.cxx` calls these functions and organizes their overall process in `Physics()`. `I3DeepCoreHit.h` defines a struct used in the other files to hold DOM hit information.

A.2 Parameters

This module has a total of nine configurable parameters, as defined in `I3DeepCoreVeto.cxx`. They are:

1. `I3MCTreeName`, a string containing the name of the monte carlo tree to get from the frame. Takes a string.
2. `InputDeepCoreRecoHitSeries`, a string containing the name of the hit series which includes reco hits only for the event's *Deep Core* DOMs.
3. `InputDeepCoreRecoPulseSeries`, a string containing the name of the pulse series information for the same set of DOMs as above.
4. `InputIceCubeRecoHitSeries`, a string containing the name of the hit series which includes reco hits for all DOMs *except* those in *DeepCore*.
5. `InputIceCubeRecoPulseSeries`, as above, a string containing the name of the pulse series that contains information for all DOMs *outside* of *DeepCore*.

6. `DecisionName`, a string containing the name given to the boolean pushed to the frame which holds information about how many events were/were not vetoed.
7. `RecordNonDCEvents`, a bool asking if you would like to record into the decision any information about events that did NOT trigger any DeepCore DOMs (True = yes).
8. `RecordVetoedDCEvents`, a bool asking if you would like to record into the decision any information about events that triggered Deep Core but were subsequently vetoed due to coincident hits. (True = yes).
9. `RecordKeptDCEvents`, a bool asking if you would like to record to the frame any information about the decision and other calculated values for events that triggered DeepCore but did not have enough (or any) coincident hits to be discarded. (True = yes).

Note: `InputDeepCoreRecoHitSeries` and `InputIceCubeRecoHitSeries` should not have any overlapping DOM information; they should define the DOMs as describe earlier in 1. The same applies to the respective pulse series.

A.3 Further Module Controls

Other aspects of the algorithm may also be easily fine-tuned.

- The veto window of values associated with particle speed distribution used to determine if an icecube hit is coincident with DeepCore or not. Possible options, as determined from optimization studies utilizing signal and corsika MC include 0.25–0.40 or 0.25–0.45 meters per nanosecond, inclusive.
- The number of icecube hits to allow to fall in this window, before vetoing an event.

B Coding

B.1 Computing the mean and standard deviation of an event's DeepCore hit times

```
//
// iterator: selector
// Record and average the DeepCore hit times.
//
double sumDCTime = 0.0;
double avgDCTime = 0.0;
I3RecoPulseSeriesMap::const_iterator selector;
for(selector=hitmap->begin(); selector!=hitmap->end(); selector++)
{
    const I3RecoPulseSeries& hits = selector->second;
    // make sure the pulse series is not empty
    if(!hits.size()) log_error("DeepCore pulse series is empty!");

    // only use first pulse for hit time
    const I3RecoPulse& pulse = hits.front();
    I3DeepCoreHit hit;
    hit.hitTime_ = pulse.GetTime();

    // Check for Nan and INF in the timing
    if ( isnan(hit.hitTime_)==1 || isinf(hit.hitTime_)==1 )
{
    // Don't Add the hit to the list
    log_info("DEEPCORE: Got Nan or INF for hit Time! not recording
            this hit");
    continue;
}

    // If info exists correctly...
    else
{
    sumDCTime += pulse.GetTime();
    deepCoreHits_->push_back(hit);
    log_debug("Got time %f ",hit.hitTime_);
}
}
```

```

    } // end for loop - selector.

// Calculate the mean time
double numDCHits = GetNumDeepCoreHits();
avgDCTime = sumDCTime / numDCHits;

//
// iterator: iter1
// Calculate the standard deviation for the distribution of
// DeepCore hit times.
//
double sum_delta_squared = 0.0;
double std_deviation = 0.0;
I3DeepCoreHitSeries::const_iterator iter1;
for(iter1 = deepCoreHits_->begin(); iter1 != deepCoreHits_->end();
    iter1++)
{
    const I3DeepCoreHit& hit = *iter1;
    double hitTime = hit.hitTime_;
    sum_delta_squared += ( (hitTime - avgDCTime)*
                          (hitTime - avgDCTime) );
}
std_deviation = sqrt( sum_delta_squared / (numDCHits-1) );

```

B.2 Calculating the DeepCore center of gravity and associated time

```

//
// iter2
// Calculate the center of gravity
//
double sumWeights = 0.0;
double charge = 0.0;
double sumX = 0.0; double sumY = 0.0; double sumZ = 0.0;
unsigned calccoghitNum = 0;
I3DeepCoreHitSeries::const_iterator iter2;
for(iter2 = deepCoreHits_->begin(); iter2 != deepCoreHits_->end();

```

```

iter2++)

{
    const I3DeepCoreHit& hit = *iter2;

    // Get the charge of this hit and use it to find the COG weight
    charge = hit.charge_;
    double weight = CalcCogWeight(charge);

    // Sum the weighted positions
    sumX += weight * hit.hitX_;
    sumY += weight * hit.hitY_;
    sumZ += weight * hit.hitZ_;
    sumWeights += weight;
    calccoghitNum++;

    log_trace("x = %.3f, y = %.3f, z = %.3f, t = %.3f, hitnum: %d",
              hit.hitX_, hit.hitY_, hit.hitZ_, hit.hitTime_, calccoghitNum);
} // for each hit

// Check that there are hits and that the sum of their weights is nonzero...
if (calccoghitNum == 0 || sumWeights == 0.0)
{
    log_error("Number of hits=%d, sumWeights=%lf",
              calccoghitNum, sumWeights);
    return false;
}
else
{
    // Calculate the Center of Gravity
    double cogX = (sumX / sumWeights);
    double cogY = (sumY / sumWeights);
    double cogZ = (sumZ / sumWeights);

    //
    // iter3
    // Calculate the average time associated with the COG
    //
    double dX = 0.0; double dY = 0.0; double dZ = 0.0;

```

```

        double dist = 0.0;
        double sumT = 0.0;
        unsigned calctimehitNum = 0;
        I3DeepCoreHitSeries::const_iterator iter3;
        for(iter3 = deepCoreHits_->begin(); iter3 != deepCoreHits_->end();
                                                    iter3++)
    {
        const I3DeepCoreHit& hit1 = *iter3;

        // Trace each hit back to the COG
        dX = hit1.hitX_ - cogX;
        dY = hit1.hitY_ - cogY;
        dZ = hit1.hitZ_ - cogZ;
        dist = sqrt(dX*dX + dY*dY + dZ*dZ);
        // Sum the corrected times
        sumT += abs( (hit1.hitTime_-(dist /
                                    (I3Constants::c/I3Constants::n_ice))) );
        calctimehitNum++;

        log_trace("dX=%lf, dY=%lf, dZ=%lf, sumT=%lf, hit# %d",
                  dX,dY,dZ,sumT,calctimehitNum);
    }

    // Set the Mean Corrected Time
    vertexTime = (sumT / calccoghitNum);

    // Set the center of gravity
    CoG.SetPosition(cogX, cogY, cogZ);
}

```

B.3 Calculating the distance / time difference

```

/* In a loop over all IceCube hits... */

// Create an I3Position to hold the current icecube hit
I3Position HitPos(icecube_hitX, icecube_hitY, icecube_hitZ);

// Calculate the distance and time delay between the icecube

```



```

//hit and the (previously calculated) deepcore Center of Gravity
distance = sqrt( (CoG.CalcDistance(HitPos))*
                 (CoG.CalcDistance(HitPos)) );
time_difference = (vertexTime - icecube_hitTime);

//
// Particle_speed represents the speed at which a particle
// would have to travel to make it from the icecube hit to
// the calculated COG in the time difference between those 'hits'
// It is positive if the icecube hit occurred before CoG time
// (vertexTime), negative if it occurred after
//
particle_speed = (distance / time_difference);

```

B.4 Checking if distance / time difference is in the veto window

```

/* Also in a loop over all IceCube hits... */

// Check if the particle speed for this icecube hit is
// in the veto window
nicecubeHits++;
InWindow(particle_speed, nVetoWindowHits);

// If the event has not been vetoed (discarded) yet, check the
// number of hits to allow in the veto window.
// If that limit has been exceeded, permanently discard the event
if (veto=true && nVetoWindowHits <= 1)
{
    log_debug("Event not vetoed in I3DeepCoreFitter::Veto()");
    veto = true;
}
else if (veto=true && nVetoWindowHits > 1)
{
    log_debug("Event vetoed in I3DeepCoreFitter::Veto()");
    veto = false;
}

```

```
else    // If the event is already discarded - if (veto==false)...  
    continue;
```