

LArIAT Aerogel Module

Brandon Soubasis, Dung Phan and Will Flanagan
The University of Texas at Austin

May 2nd, 2016

1 Introduction

Aerogel counter is used to differentiate pions from muons in the tertiary beam. The Cherenkov light produced when the charged particles travel through the aerogel will be collected by the PMTs; and the waveforms will be recorded for later analysis.

Aerogel information is analyzed by AGCModule in LArIATSoft framework. This module contains three classes:

- Located in /LArIATRecoModule, AerogelCherenkovCounterSlicing class collects the needed waveforms in the form of the AuxDetDigit, feed them to the AGCounterAlg class for needed calculations; and finally send the packaged results to the AGCounter data product.
- Located in /LArIATRecoAlg/, AGCounterAlg class is the most important one in this module. It needs to be fed with a set of four waveforms (2 from KEK aerogel counter, 2 from UTKL aerogel counter). It would return 4 booleans, which tells which of the 4 PMTs fired up, 4 doubles, which are the pulse areas recorded if the corresponding PMT is fired up, and other 4 doubles, which are the timestamps of the hits. This module will be explained in details in the next section.
- Located in /LArIATDataProduct/, AGCounter class will package the processed information and save them to an appropriate ROOT file.

2 Overview on AGCounterAlg

AGCounterAlg contains the following PUBLIC function members:

- ImportWaveform(std::string DetName, std::vector<raw::AuxDetDigit>): This function is called when you want to import a waveform. For each event, this should be called 4 times (4 PMTs). The **DetName** variable can be: "USE" for upstream-east PMT, "USW" for upstream-west PMT, "DS1" for downstream-first PMT, "DS2" for downstream-second PMT.
- AGCHitsWrapper(): This function will package and return a vector of **AGCHits** structure. Each of this structure contains:
 - TriggerTimeStamp: The timestamp of the event.
 - HitTimeStampUSE, HitTimeStampUSW, HitTimeStampDS1, HitTimeStampDS2: The timestamps of the hits in the four PMTs.
 - HitPulseAreaUSE, HitPulseAreaUSW, HitPulseAreaDS1, HitPulseAreaDS2: Pulse areas of the waveforms from 4 PMTS created by the hit.

- HitExistUSE, HitExistUSW, HitExistDS1, HitExistDS2: A hit is able to trigger one PMT but not others. So this variable tells us which in four PMTs is fired up when there is a hit.

AGCounterAlg contains the following PRIVATE function members:

- `std::vector<int> HitsFinder(raw::AuxDetDigit)`: You feed this function with a waveform, it will return a vector of integers, which are the ADC ticks at which there is a hit.
- `float PulseAreaFinder(raw::AuxDetDigit, long unsigned int)`: This function need to be fed with a waveform and an integer that tells the ADC tick of the hit. The function then calculate the pulse area of that hit by integrating the waveform in an appropriate time window.
- Two other functions, `CheckMatched()` and `HitsTimeMatching()`, are related to the algorithm to form hit groups so we will talk about them later.

2.1 Hit finder algorithm

```

1  std::vector<int> AGCounterAlg::HitsFinder(raw::AuxDetDigit WaveformDigit) {
2      float Threshold = -40;
3      std::vector<int> Hits;
4      Hits.clear();
5      bool RisingEdge = false;
6      for (size_t i = 2; i < WaveformDigit.NADC()-2; ++i) {
7          // Five-point stencil derivative
8          float Derivative = float(8*(WaveformDigit.ADC(i+1))-8*(WaveformDigit.ADC(i-1))
9                               +WaveformDigit.ADC(i-2)-WaveformDigit.ADC(i+2))/12;
10         if ((Derivative < Threshold)&&(RisingEdge == false)) {
11             Hits.push_back(i);
12             RisingEdge = true;
13         }
14         if ((Derivative > std::abs(Threshold))&&(RisingEdge == true)) {
15             RisingEdge = false;
16         }
17     }
18     return Hits;
19 }
```

When we keep scanning through the waveform, we need a flag to tell if we have already pass a hit and being on the way to find a new hit. This is done by setting the flag "RisingEdge." When we are looking for a hit, the RisingEdge flag will be set to "false" (in line [4], before the loop, we set the RisingEdge to false, and in line [14] we set this flag back to false after we passed the hit). Right after we found a hit then this flag is set to true, as in line [11].

Now, we scan through the waveform (line [5], we loop over the time ticks of the waveform), and calculate the derivative of the waveform (line [7], using the method of five-point stencil), which is also the slope of the waveform curve. If, at some time tick, the slope is larger than a pre-defined threshold (chosen and hardcoded to be -40 ¹, line [1], the threshold is negative because the signals from the PMTs are negative-polarized), and the "RisingEdge" flag is false, then we decide that there is a hit at that time tick.

¹This mean the rising edge must be very steep to be consider a hit

2.2 HitsTimeMatching algorithm

This function is used to group the hits in different PMTs into hit groups. The input for this function is 4 vectors of integers that indicates the time ticks of hits in 4 waveforms, line [1]. The function will return a vector of a set of 4 integers (a vector with 4 elements), line [1]. The index of the outer vector tells us how many hits are there in the event.

```
1 std::vector<std::vector<int> > AGCounterAlg::HitsTimeMatching(std::vector<int> USEHits,
2                                                                std::vector<int> USWHits,
3                                                                std::vector<int> DS1Hits,
4                                                                std::vector<int> DS2Hits) {
5     std::vector<std::vector<int> > AllHitTimePairing;
6     std::vector<int> HitTimePairing;
7     for (size_t i = 0; i < USEHits.size(); i++) {
8         for (size_t j = 0; j < USWHits.size(); j++) {
9             for (size_t k = 0; k < DS1Hits.size(); k++) {
10                for (size_t l = 0; l < DS2Hits.size(); l++) {
11                    HitTimePairing.push_back(USEHits.at(i));
12                    HitTimePairing.push_back(USWHits.at(j));
13                    HitTimePairing.push_back(DS1Hits.at(k));
14                    HitTimePairing.push_back(DS2Hits.at(l));
15                    if (CheckMatched(HitTimePairing)) {
16                        AllHitTimePairing.push_back(HitTimePairing);
17                    }
18                    HitTimePairing.clear();
19                }
20            }
21        }
22    }
23    return AllHitTimePairing;
24 }
```

In line [7, 8, 9, 10], we loop over the vectors of integers (time ticks of hits) from all four PMTs. We don't need to focus too much on the "HitTimePairing" vector, it's just a convenient way to transfer the argument to the "CheckMatched()" function. The CheckMatched() function will then decide if the input integers actually formed a hit group, line [15]. If a hit group is found, it will be saved to "AllHitTimePairing" vector, line [16].

The "CheckMatched()" decide if a set of four hit time ticks from 4 PMTs formed a hit group by comparing the time differences between the time ticks.

```
1 bool AGCounterAlg::CheckMatched(std::vector<int> HitTimePairing) {
2     short TimeThreshold = 15;
3     std::vector<int> TimeMetrics;
4     for (size_t i = 0; i < HitTimePairing.size(); i++) {
5         if (HitTimePairing.at(i) != 0) {
6             TimeMetrics.push_back(HitTimePairing.at(i));
7         }
8     }
9     bool FormedHitGroup = true;
10    for (size_t i = 0; i < TimeMetrics.size(); i++) {
11        for (size_t j = i+1; j < TimeMetrics.size(); j++) {
12            if (std::abs(TimeMetrics.at(i)-TimeMetrics.at(j)) > TimeThreshold) {
```

```
13         FormedHitGroup = false;
14         j = TimeMetrics.size();
15         i = j;
16     }
17 }
18 }
19 return FormedHitGroup;
20 }
```

If among the 4 input hit time ticks, the difference between any two is larger than a time difference threshold then the boolean "FormedHitGroup" will be set to "false."