

THE EFFECT OF PHOTOELECTRON LEAKAGE AT THE FACE OF THE
PMT ON SCINTILLATOR EFFICIENCY STUDIES OF COSMIC RAY VETO

Jefferson Lansford
Undergraduate, Physics
University of Virginia
08 May 2011

ABSTRACT

Initial analysis of the Summer 2010 Half Module Data for the Cosmic Ray Veto (CRV) system indicates that leakage of photoelectrons (PE) at the face of the photo multiplier tubes (PMTs) may be influencing calculated scintillator efficiencies. This investigation first examines the evidence for leakage of PE at the face of each PMT and then attempts to ascribe functional associations to the relationship between leakage and calculated scintillator efficiency. To examine the evidence for leakage, we plotted histograms of the response for each scintillator (given a vertical cosmic ray through the trigger scintillators) and ordered the histograms to reflect fiber layout at the PMT. We demonstrate that the leakage, which is similar for both PMTs, is on the order of a single photoelectron (SPE) and that it is typically constrained to horizontally and vertically adjacent fibers on the PMT. We then go on to describe the relationship between leakage and calculated scintillator efficiency. By increasing the thresholds of the trigger scintillators, we increase the likelihood that a cosmic ray did indeed travel vertically through a specific column of the scintillators, thus providing a check to see if leakage causes calculated single layer efficiency (SLE) to be too low. By increasing the thresholds of the middle scintillator, we increase the likelihood that the middle scintillator was not above threshold because of leakage, thus providing a check to see if leakage causes calculated SLE to be too high. We determine that the former effect is indeed observable but that increasing errors that arise with increasing trigger thresholds prevent statistically significant results. However, a theoretical 'leakless' SLE of .988 was calculated but again with a less than satisfying error.

TABLE OF CONTENTS

1. Background
 - 1.1. Conservation of Lepton Number
 - 1.1.1. Known Path for Muon Decay
 - 1.1.2. Predicted Path for Muon Decay
 - 1.2. Mu2e Experiment
 - 1.2.1. Overview
 - 1.2.2. Cosmic Ray Veto
2. Half Module
 - 2.1. Schematics
 - 2.1.1. Scintillator and PMT Numbering
 - 2.2. Summer 2010
 - 2.2.1. Data Summary
 - 2.2.2. Trigger Placement
 - 2.2.3. Initial Analysis
3. Data Processing
 - 3.1. Evidence for Leakage
 - 3.1.1. Processing Methodology
 - 3.1.2. Output
 - 3.2. Leakage and Efficiency
 - 3.2.1. Processing Methodology
 - 3.2.2. Output
4. Trends and Conclusions
 - 4.1. Evidence for Leakage
 - 4.1.1. Understanding the Trend
 - 4.1.2. 'Left' vs 'Right' Side of Apparatus
 - 4.2. Leakage and Efficiency
 - 4.2.1. Understanding the Trend
 - 4.2.2. Statistical Uncertainty
 - 4.3. Recommendations for Future Study
5. References and Appendices
 - 5.1. Works Cited
 - 5.2. Thresholds and SPE Peaks
 - 5.3. Data Processing Algorithms
 - 5.3.1. Evidence for Leakage
 - 5.3.2. Leakage and Efficiency

BACKGROUND

1.1 Conservation of Lepton Number

The six leptons in the Standard Model are the charged electron, muon, tau particle and each of their neutral ‘associated’ neutrinos. These elementary particle ‘pairs’ are distinct in that they each have a different quantum number for ‘flavor.’ Conservation of lepton number is one of the fundamental tenets of the Standard Model. Furthermore, conservation of lepton number typically includes the stipulation that ‘flavor’ be conserved as well¹. Roughly, this means that in the following decays, c+d must be equal to a+b:

$$a \bullet e + b \bullet \nu_e + other \rightarrow c \bullet e + d \bullet \nu_e + other$$

$$a \bullet \mu + b \bullet \nu_\mu + other \rightarrow c \bullet \mu + d \bullet \nu_\mu + other$$

$$a \bullet \tau + b \bullet \nu_\tau + other \rightarrow c \bullet \tau + d \bullet \nu_\tau + other$$

1.1.1 Known Path for Muon Decay

The muon and tau particle are not stable under everyday conditions. Typically, muons undergo Michel-decay in the presence of a lighter nucleus such that:

$$\mu^- + (A, Z) \rightarrow e^- + \bar{\nu}_e + \nu_\mu + (A, Z)$$

This conserves lepton number and flavor because the anti-(electron-neutrino) ‘cancels’ the lepton number of one for the electron so that each side of the decay has a lepton number of one (both of ‘flavor’ muon). The same is also true for the typical ‘capture’ of a muon by a heavier nucleus such that²:

$$\mu^- + (A, Z) \rightarrow \nu_\mu + (A, Z - 1)$$

1.1.2 Predicted Path for Muon Decay

The Standard Model does not strictly disallow lepton flavor violation (i.e. processes that do not conserve lepton flavor). These processes have in fact been observed in the form of neutrino oscillations. However, according to the Standard Model, charged-lepton flavor violation (CLFV) should be dynamically suppressed to a practically immeasurable level due to neutrino interactions. However, there exist ‘Beyond’ Standard Model theories which are in accord with observed neutrino masses and which predict that the following rare decays occur at measurable levels³:

$$\mu \rightarrow e\gamma$$

$$\mu \rightarrow eee$$

$$\mu \rightarrow e$$

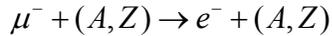
¹ *Hyperphysics*. Quantum Physics, Particles, Particle Conservation Laws

² “Proposal,” 17

³ Marciano, 317

1.2 Mu2e Experiment

The Mu2e Experiment attempts to observe CLFV in the form of direct conversion of a muon to an electron in presence of a nucleus such that⁴:



Observing this decay would provide support for the ‘Beyond’ Standard Model theories that predict it while failing to observe the decay with the greater precision of the Mu2e experiment would place constraints on these theories.

1.2.1 Overview

Currently, the ratio of direct conversion events to muon capture events in the presence of a gold nucleus is known to be less than $6.1 \cdot 10^{-13}$ at the 90% confidence level.⁵ Over a two-year run, the Mu2e Experiment should be able to either lower that ratio to an order of $\sim 10^{-17}$ or observe direct conversion of a muon to an electron at the 90% confidence level.⁶

1.2.2 Cosmic Ray Veto

Cosmic rays striking the upper atmosphere produce a cascade of lighter particles in high energy collisions so that approximately half of cosmic radiation at sea level is in the form of muons.⁷ Without passive and active shielding, these muons would create an unwanted background of electrons at the energy expected for direct muon to electron conversion by scattering electrons from the target and by in-flight Michel-decay into electrons, thereby destroying the potential sensitivity of the experiment.⁸

This background can be reduced to a non-interfering level with a combination of active and passive shielding. The passive shielding is to be accomplished by building the experimental apparatus underground and by encasing the detector in 2.0m of concrete and 0.5m of steel. The active shield will disregard results that coincide with passing cosmic rays which will be detected with 99.99 percent efficiency as they pass through a triple layer of scintillating material that detects charged particles (and thus cosmic ray muons and electrons).⁹ This active shield is called the cosmic ray veto (CRV) system.

HALF MODULE

2.1 Schematics

To achieve a 99.99% efficiency for CRV, each layer in the triple layer of scintillating material would need to have an efficiency of at least 99%. In the summer of 2009 and 2010, a model of the CRV underwent preliminary testing. This model was constructed using 30 scintillators of dimension 470cm x 10cm x 1cm where these bars were layered and numbered via the diagram

⁴ “Proposal,” 5

⁵ SI Collaboration, 337

⁶ “Proposal,” 20

⁷ *Hyperphysics*. Astrophysics, Cosmic Rays

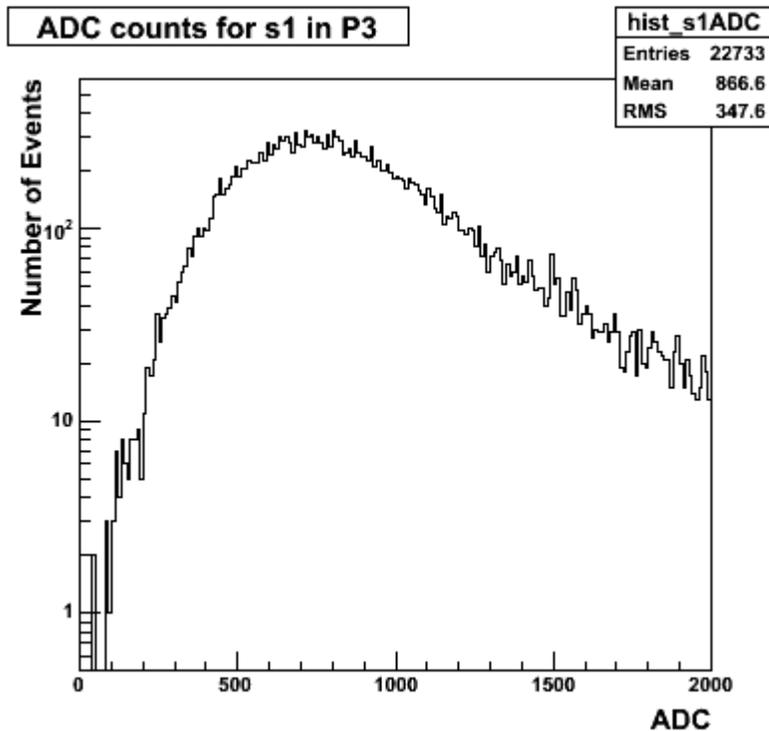
⁸ “Proposal,” 32

⁹ “Proposal,” 51

2.2.2 Trigger Placement

The P0 run was a calibration run taken with the trigger placed away from the scintillator apparatus. For the P1-P7 runs, the trigger was placed perpendicularly to the long axes of the scintillator bars at varying distances from the readout end. P1 was 50cm from the readout end and each successive P_ run was an additional 50cm from the readout end.¹²

Furthermore, for the summer 2010 data, the second trigger was placed below the scintillator apparatus and a third trigger was also used. This configuration reduced the percentage of ‘hot’ events from about 8% to 0.5%.¹³ Based on the data in runs P1-P7, this trigger was connected to the channel on the PMT that the diagram above indicates as corresponding to s1. The plot below is from P3 and is representative of P1-P7.



2.2.3 Initial Analysis

In Mu2e-doc-1382, the single layer efficiency (SLE) for the middle layer of scintillators was calculated to be 91-94% depending on position from the readout end (for the 2010 data). This calculation is taken to be performed as follows for each run:

$$\varepsilon_n = \frac{\text{Number_of_events}[sn \ \& \ s1n \ \& \ s2n \ > \ \text{respective_thresholds}]}{\text{Number_of_events}[sn \ \& \ s2n \ > \ \text{respective_thresholds}]}$$

$$\varepsilon = \sum_{n=1}^{10} \varepsilon_n$$

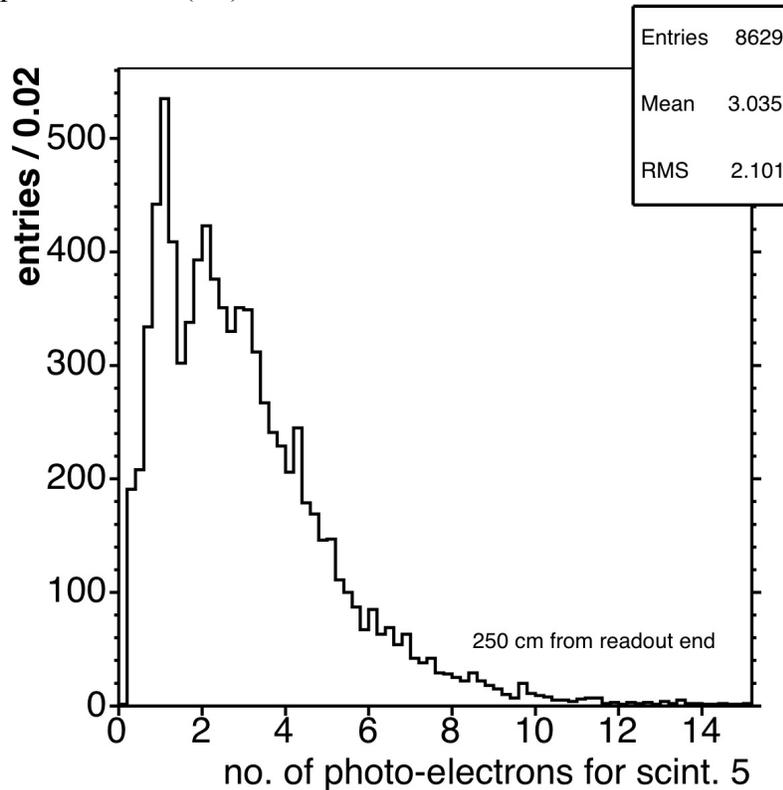
¹² Glenzinski 777, 4

¹³ Glenzinski 1382, 66

where ‘n’ designates a particular ‘column’ of three scintillators. These efficiencies are raised by 1-2% if the offset of the middle scintillator is taken into effect. This is done with an ‘or’ statement in the numerator to include the possibility that the cosmic ray went through the nearest neighbors¹⁴. The numerator would then become:

$$\text{Number_of_events}[s_n \& \{s_1(n-1) \parallel s_1n \parallel s_1(n+1)\} \& s_2n > \text{respective_thresholds}]$$

It was also noted in document 1382 that the histograms for the above threshold events in each scintillator do not follow Poisson statistics but rather have an excess at what is assumed to be 1 photoelectron (PE) as shown below¹⁵



It was suggested that efficiencies be reinvestigated by raising the thresholds for the denominator to >1.7PE to remove the 1PE noise peak from the top/bottom scintillators.¹⁶

¹⁴ Glenzinski 1382, 43

¹⁵ Glenzinski 1382, 55

¹⁶ Glenzinski 1382, 61

DATA PROCESSING

3.1 Evidence for Leakage

This section will present data that will hopefully clarify the nature of what we mean by ‘leakage’ in this study.

3.1.1 Processing Methodology

To develop the methodology for characterizing leakage at the face of the PMTs, it is first necessary to examine the fiber schematics of Section 2.1.1 which detail the connections between the scintillator apparatus and the PMTs. The fibers of two scintillators can have four distinct relationships at the face of the PMTs: (I) The fibers are horizontally/vertically adjacent on the same PMT. (II) The fibers are diagonally adjacent on the same PMT. (III) The fibers are not adjacent but are attached to the same PMT. (IV) The fibers are not attached to the same PMT.

The leakage will first be examined by inspection. This will be done by making histograms for all the scintillators with the condition that s5, s15, and s25 be above threshold.¹⁷ The histograms will be arranged to reflect their layout on the PMTs. This process will be repeated with the condition that s6, s16, and s26 be above threshold.

The fiber schematics will also be exploited to derive a numerical description of ‘leakage.’ For each PMT, two dimensional histograms will catalogue the combined above threshold response (in number of photoelectrons) for all scintillators with a Type I relation to the trigger scintillators described above. These values are plotted against the average number of photoelectrons in the triggers. This will be repeated for the combined Type II/III relations and the Type IV relations.

The following programs in C++ (attached as appendices) were written to effect each of the above:

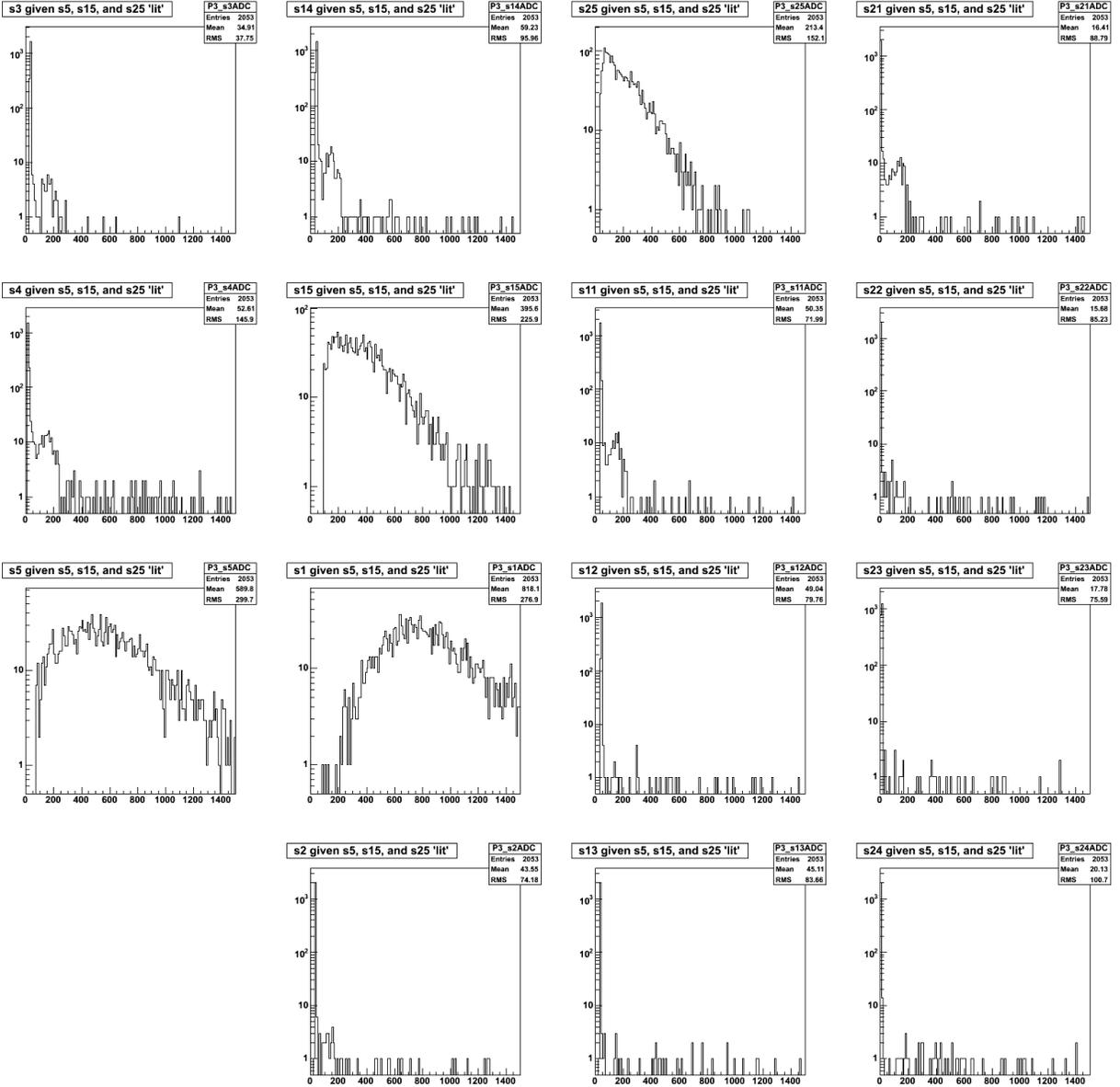
PMT_chart.C
Leakage_LR.C

¹⁷ Thresholds and SPE peaks discussed in Appendix A

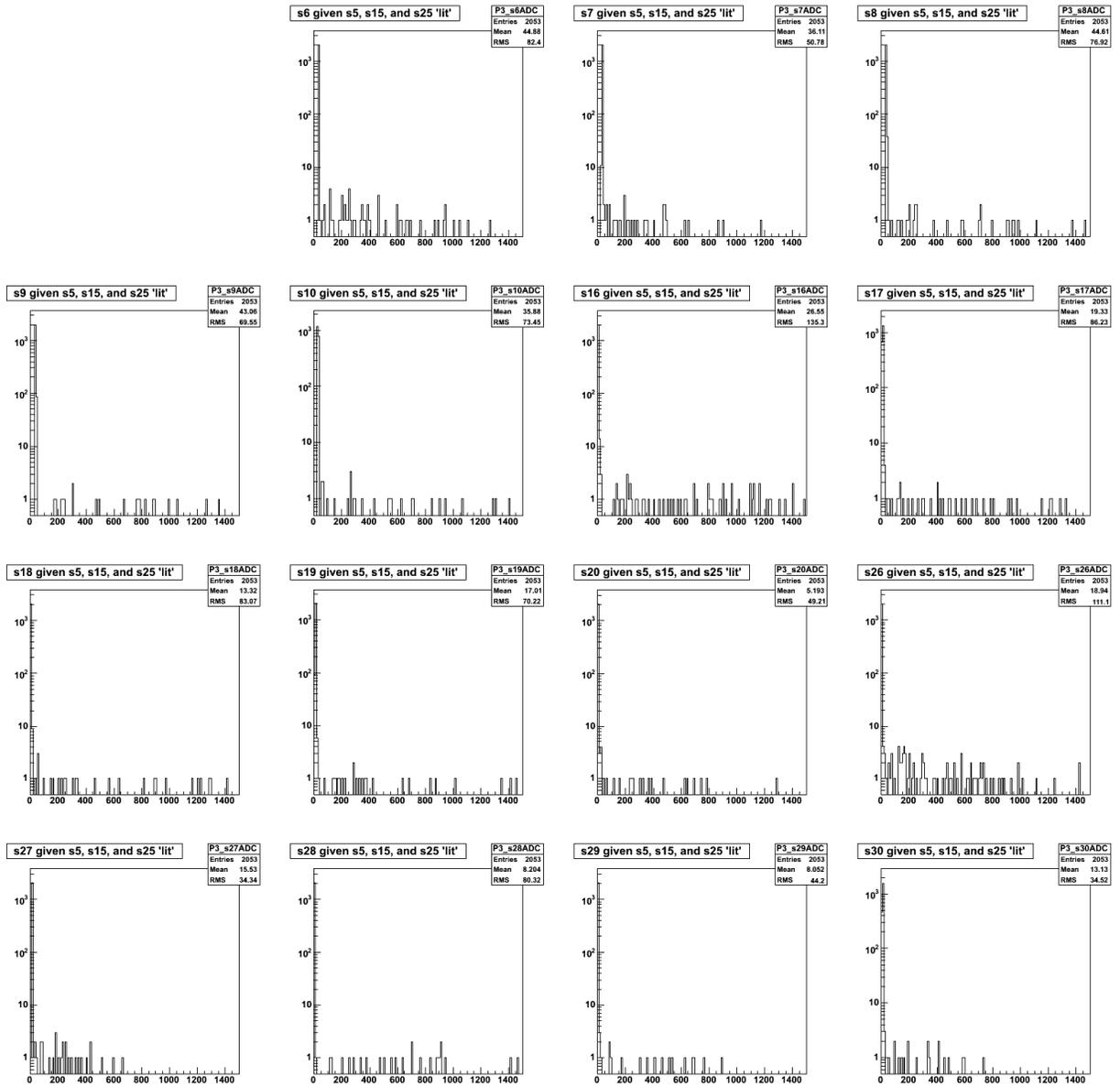
3.1.2 Output

Plots 'a' through 'd' were constructed with PMT_chart.C

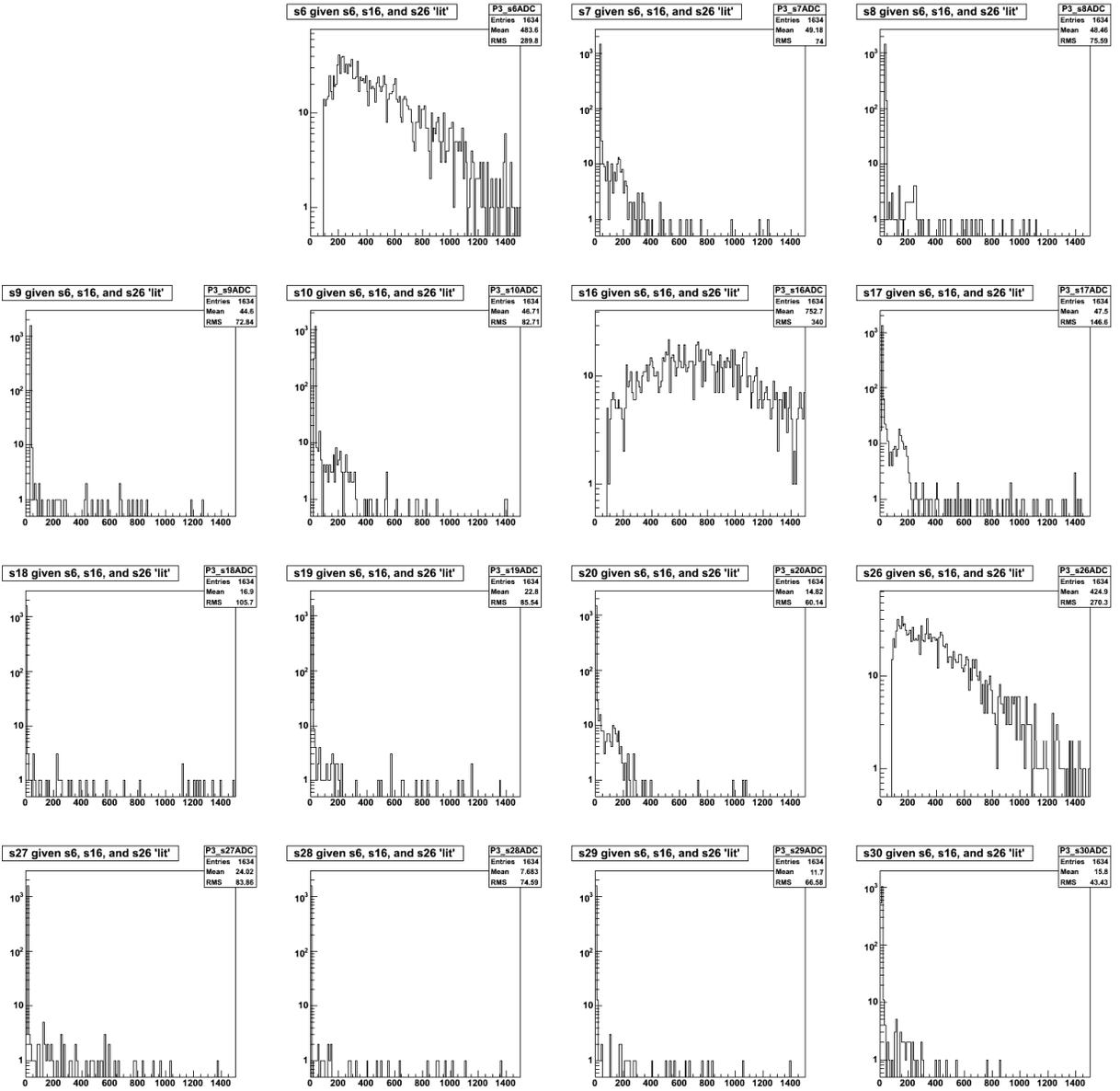
a. 'Right' PMT ('left' side of apparatus when viewed from the readout end) for s5, s15, s25 above threshold



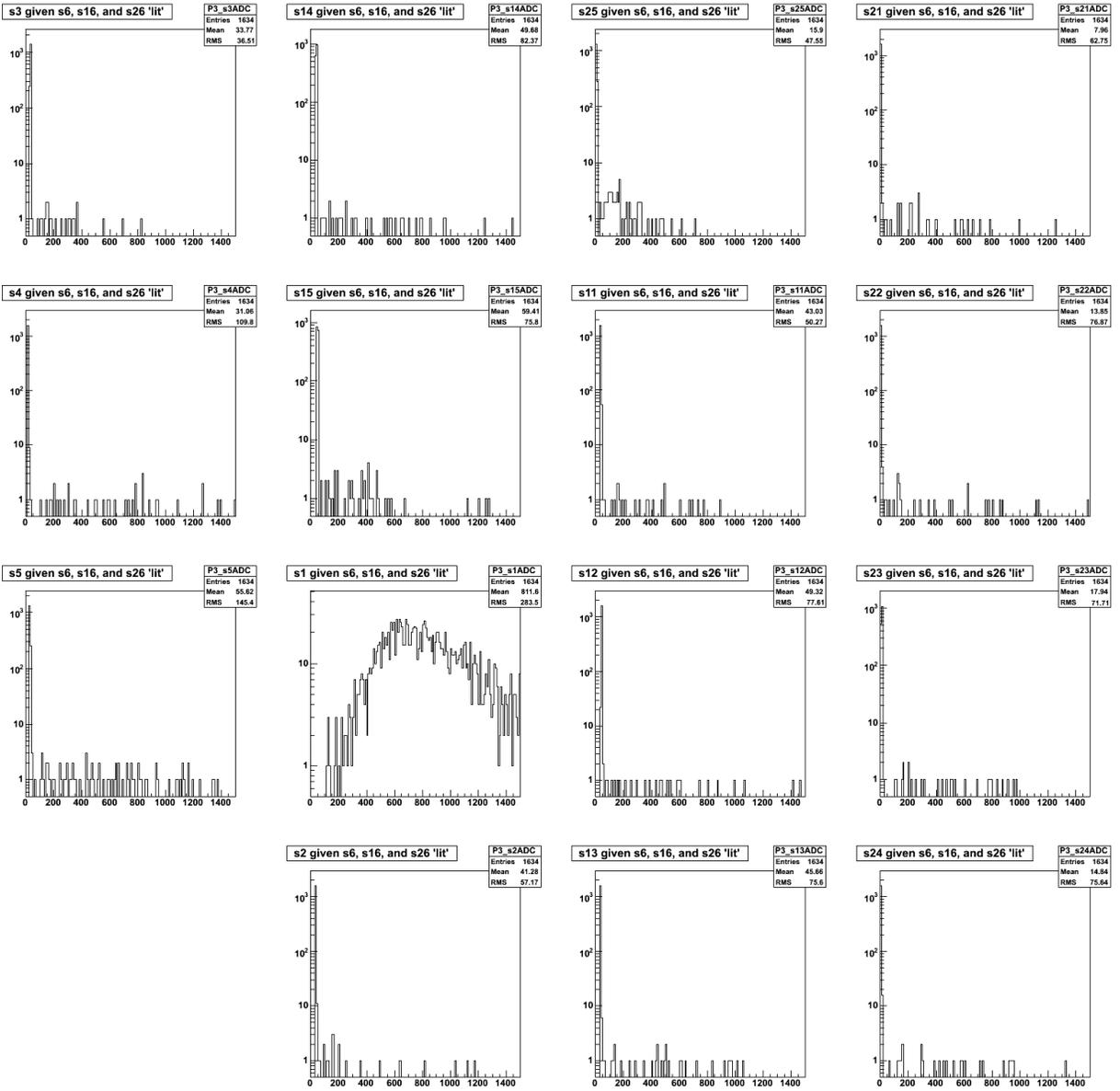
b. 'Left' PMT ('right' side of the apparatus when viewed from the readout end) for s5, s15, s25 above threshold



c. 'Left' PMT for s6, s16, s26 above threshold



d. 'Right' PMT for s6, s16, s26 above threshold



e. Text Output for Leakage_LR.C

PMT 0 corresponds to the right PMT and thus the left half of the apparatus

-The triggers for PMT 0 are s5, s15 and s25

-The Type I channels are s4, s11, s14 and s21

-The Type II/III channels are s2, s3, s12, s13, s22, s23 and s24

-The Type IV channels are the channels attaching to the other PMT, excluding s10 because this is the equivalent of excluding s1 on PMT1 (which is used for a trigger)

PMT 1 corresponds to the left PMT and thus the right half of the apparatus

-The triggers for PMT 1 are s6, s16 and s26

-The Type I channels are s7, s17, s20 and s30

-The Type II/III channels are s8, s9, s18, s19, s27, s28 and s29

-The Type IV channels are the channels attaching to the other PMT, excluding s1 (which is used for a trigger)

For all calculations, 'exp prob' stands for 'experimental probability' and is calculated such that:

$$P = \frac{\sum_{\text{events}} \# \text{scin_of_TypeX_above_threshold_given_}(triggers > threshold)}{\# \text{events_with_all_three_triggers_above_threshold}} \bigg/ \# \text{scin_of_TypeX}$$

22733 events in P3 with 22513 events not 'hot' (a 'hot' event is defined to be any event with > 15 scintillators above threshold and 'hot' events are excluded from the results)

0.0903034 percent of events for PMT 0 with all three trigger scintillators above threshold (2033 number of such events)

0.0807919 TypeI leakage exp prob for PMT 0
657 number of such events (4 scint per 'event')

0.0210105 TypeII/III leakage exp prob for PMT 0
299 number of such events (7 scint per 'event')

0.0169349 TypeIV leakage exp prob for PMT 0
482 number of such events (14 scint per 'event')

0.0720028 percent of events for PMT 1 with all three trigger scintillators above threshold (1621 number of such events)

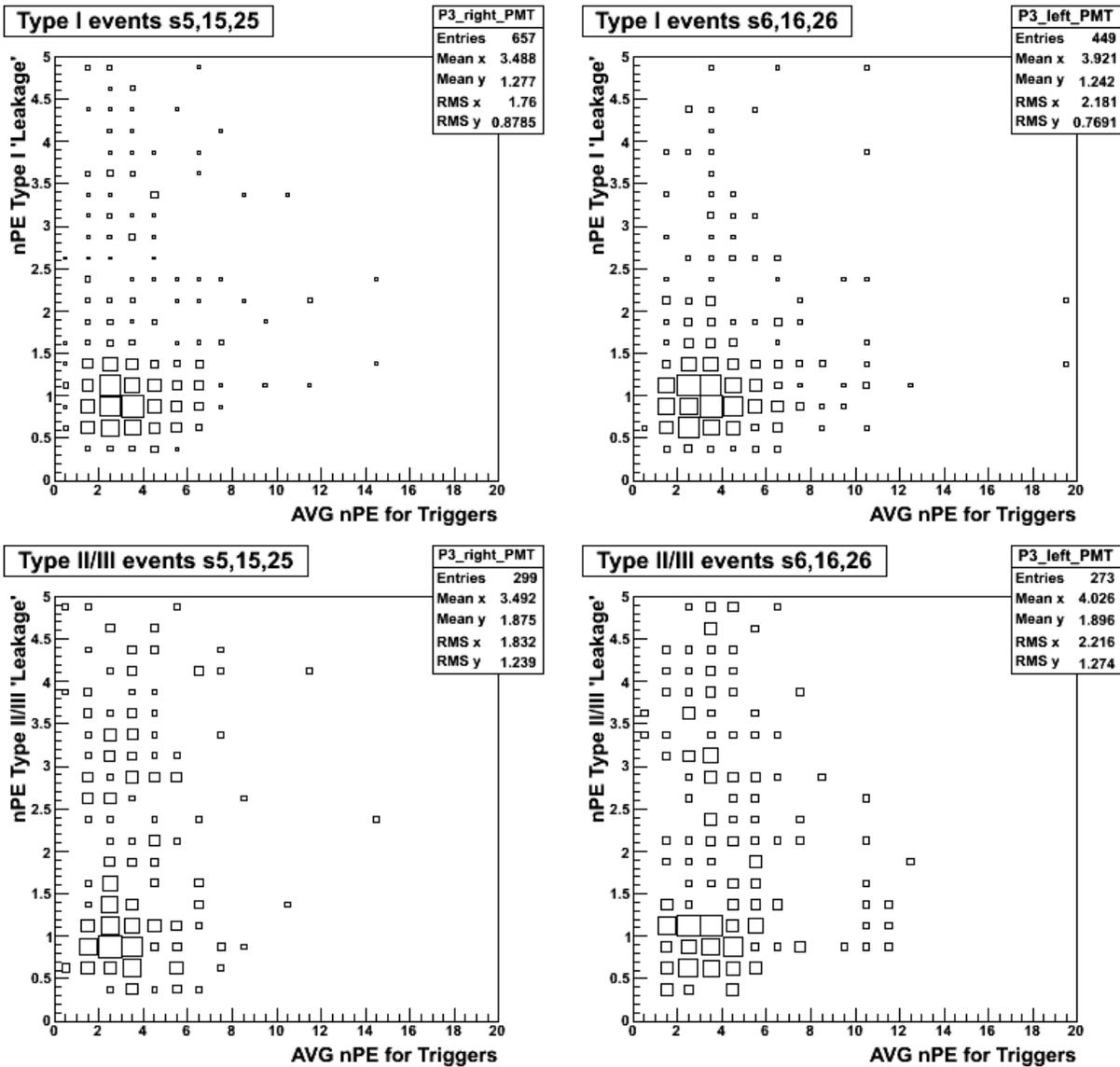
0.0692474 TypeI leakage exp prob for PMT 1
449 number of such events (4 possible scint per 'event')

0.0240592 TypeII/III leakage exp prob for PMT 1
273 number of such events (7 possible scint per 'event')

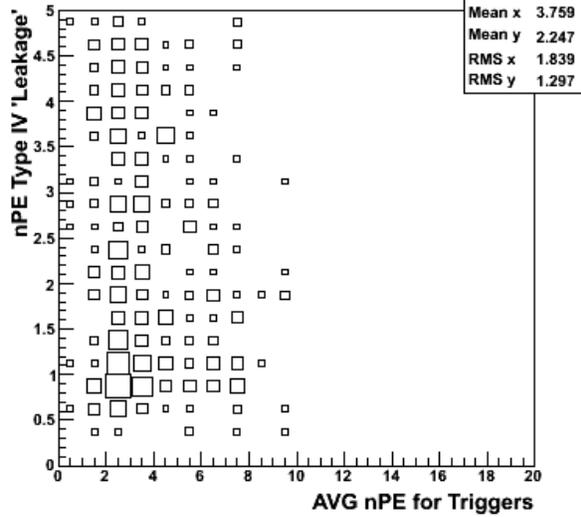
0.0220323 TypeIV leakage exp prob for PMT 1
500 number of such events (14 possible scint per 'event')

f. Graphical Output for Leakage_LR.C

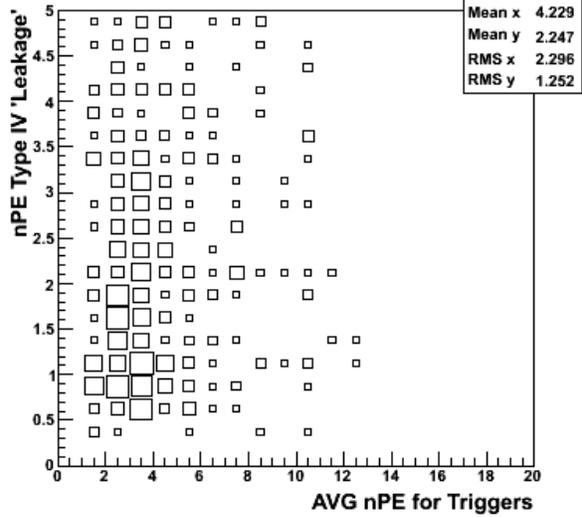
The first pair of two-dimensional histograms below plot the Type I ‘Leakage’ events (in nPE) for *all* scintillators with a Type I relation to at least one of the trigger scintillators. The second two histograms do so for the combined Type II/III events and the third two histograms do so for the Type IV events. The x-axis is an average number of PE for the three trigger scintillators and the scaling for nPE was done by dividing the measured ADC by the mean of the SPE peak. NB: Because there are a different number of scintillators for each ‘TypeX,’ the number of ‘Entries’ must be divided by the number of scintillators for that particular ‘TypeX’ to compare probability of leakage (detailed in *e*).



Type IV events s5,15,25



Type IV events s6,16,26



3.2 Leakage and Efficiency

This section will present data used to make conclusions about the effect of leakage on calculated single layer efficiency (SLE).

3.2.1 Processing Methodology

To develop the methodology for characterizing the effect of leakage on SLE, it is necessary to recognize the role that thresholds play. In the original calculation, the threshold was taken to be:

$$threshold_old = \mu_{pedestal} + 5 \times \sigma_{pedestal} + 10 ADC \cong \frac{1}{3} \times \mu_{SPE}$$

However, this threshold has two negative effects. First, if there is leakage of order 1PE, then the denominator events may not actually represent events where a cosmic ray passed vertically through the column of scintillators. Secondly, this threshold does not take into account enough of the ‘heavy’ tail of the pedestal. We therefore take the thresholds to be the minimum between the pedestal and the SPE peak¹⁸. We imaginatively call this new threshold:

$$threshold_new \cong \frac{1}{2} \mu_{SPE}$$

The ‘above threshold response’ in both trigger scintillators (*sn* and *s2n*) ensures via geometry that the cosmic ray has passed through the middle scintillator (*s1n*) whose efficiency we are calculating.¹⁹ *Threshold_new* better ensures that pedestal noise does not significantly affect the calculated efficiency.

¹⁸ Modeled SPE peaks are displayed in Appendix Section 5.2. Note that the left edge of the fitted Gaussian was chosen to correspond with the *threshold_new* for each scintillator

¹⁹ Throughout this study we will use the nearest neighbor requirement given in Section 2.2.3. This is acceptable because we are calculating single *layer* efficiency. We therefore will be calculating the efficiencies for different portions of the layer rather than for specific scintillators in the layer. When we discuss an effect for ‘*s1n*,’ it would be more precise to use the phrase ‘three scintillator portion of the middle layer centered at *s1n*’

Because this threshold is below the mean for the single photoelectron (SPE) peak and because most ‘leakage’ is of order 1PE (see Section 3.1.2.f.), we would expect noise due to leakage to affect the calculated efficiency. To investigate this effect we will vary the thresholds used in calculating the SLE as functions of the parameters for the SPE peak²⁰.

We will refer to the thresholds used for the trigger scintillators (s_n and s_{2n}) as:
threshold_a

We will refer to the thresholds for the middle scintillators (s_{1n} , $s_{1(n-1)}$ and $s_{1(n+1)}$) as:
threshold_b

First, we will vary *threshold_a* such that:

$$threshold_a = a \times \mu_{SPE} \quad \text{with } a = \{0.5, 0.75, 1.0, 1.25, 1.5, 1.75, 2.0, 2.25, 2.5, 2.75, 3.0, 3.25\}$$

$$threshold_b = threshold_new$$

We will vary these thresholds such that:

$$threshold_a = \mu_{SPE} + a \times \sigma_{SPE}$$

$$threshold_b = \mu_{SPE} + b \times \sigma_{SPE}$$

First, we take ‘a’ and ‘b’ to be the following to get an overview of the trend:

| | | | | | | | | | | | | |
|---|-------|-------|-------|-----|------|-----|------|-----|------|-----|------|-----|
| n | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
| a | -0.5 | 0.0 | 0.5 | 1.0 | 1.5 | 2.0 | 2.5 | 3.0 | 3.5 | 4.0 | 4.5 | 5.0 |
| b | -0.75 | -0.50 | -0.25 | 0.0 | 0.25 | 0.5 | 0.75 | 1.0 | 1.25 | 1.5 | 1.75 | 2.0 |

Additionally, we will vary *threshold_a* as above and calculate SLE using:
threshold_b = threshold_new

This will be done using

Then, for reasons explained in Section 3.2.2, we will take ‘a’ and ‘b’ to be:

| | | | | | | | | |
|---|--------|--------|--------|-----|-------|-------|-------|-------|
| n | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| a | -0.842 | -0.524 | -0.253 | 0.0 | 0.253 | 0.524 | 0.842 | 1.282 |
| b | -0.842 | -0.524 | -0.253 | 0.0 | 0.253 | 0.524 | 0.842 | 1.282 |

The following programs in C++ (attached as appendices) were written to effect each of the above:

SLE_threshold_a.C SLE_threshold_ab.C Surface.C

²⁰ Note that the varied thresholds for each scintillator will always be greater than or equal to *threshold_new*

3.2.2 Output

a. Output from SLE_threshold_a.C

The 'row order' for the first data block is repeated throughout

| s12 | s13 | s14 | s15 | s16 | s17 | s18 | s19 | |
|--------|--------|--------|--------|--------|--------|--------|--------|----------------|
| 0.9772 | 0.9936 | 0.9620 | 0.9086 | 0.9872 | 0.9894 | 0.9766 | 0.9693 | Efficiency |
| 2895 | 2029 | 2418 | 1882 | 1646 | 1512 | 1624 | 1171 | Denominator |
| 0.0258 | 0.0312 | 0.0279 | 0.0304 | 0.0345 | 0.0361 | 0.0345 | 0.0404 | Poisson Error |
| 0.0028 | 0.0018 | 0.0039 | 0.0066 | 0.0028 | 0.0026 | 0.0038 | 0.0050 | Binomial Error |
| 0.9775 | 0.9941 | 0.9641 | 0.9153 | 0.9873 | 0.9900 | 0.9768 | 0.9681 | |
| 2715 | 1858 | 2286 | 1452 | 1414 | 1297 | 1511 | 1097 | |
| 0.0267 | 0.0327 | 0.0288 | 0.0347 | 0.0372 | 0.0390 | 0.0357 | 0.0417 | |
| 0.0028 | 0.0018 | 0.0039 | 0.0073 | 0.0030 | 0.0028 | 0.0039 | 0.0053 | |
| 0.9774 | 0.9936 | 0.9641 | 0.9133 | 0.9888 | 0.9914 | 0.9790 | 0.9693 | |
| 2479 | 1571 | 2062 | 1176 | 1163 | 1047 | 1380 | 1009 | |
| 0.0279 | 0.0355 | 0.0303 | 0.0385 | 0.0411 | 0.0434 | 0.0375 | 0.0435 | |
| 0.0030 | 0.0020 | 0.0041 | 0.0082 | 0.0031 | 0.0029 | 0.0039 | 0.0054 | |
| 0.9793 | 0.9925 | 0.9675 | 0.9119 | 0.9891 | 0.9913 | 0.9802 | 0.9729 | |
| 2267 | 1193 | 1848 | 897 | 921 | 804 | 1214 | 887 | |
| 0.0292 | 0.0407 | 0.0321 | 0.0441 | 0.0462 | 0.0495 | 0.0400 | 0.0465 | |
| 0.0030 | 0.0025 | 0.0041 | 0.0095 | 0.0034 | 0.0033 | 0.0040 | 0.0054 | |
| 0.9792 | 0.9913 | 0.9703 | 0.9130 | 0.9905 | 0.9936 | 0.9794 | 0.9719 | |
| 2015 | 917 | 1686 | 701 | 738 | 621 | 1068 | 784 | |
| 0.0310 | 0.0464 | 0.0337 | 0.0499 | 0.0517 | 0.0565 | 0.0426 | 0.0494 | |
| 0.0032 | 0.0031 | 0.0041 | 0.0106 | 0.0036 | 0.0032 | 0.0043 | 0.0059 | |
| 0.9820 | 0.9904 | 0.9691 | 0.9157 | 0.9892 | 0.9937 | 0.9784 | 0.9710 | |
| 1726 | 729 | 1489 | 498 | 558 | 474 | 926 | 689 | |
| 0.0336 | 0.0520 | 0.0358 | 0.0593 | 0.0594 | 0.0646 | 0.0457 | 0.0527 | |
| 0.0032 | 0.0036 | 0.0045 | 0.0125 | 0.0044 | 0.0036 | 0.0048 | 0.0064 | |
| 0.9808 | 0.9929 | 0.9700 | 0.9290 | 0.9909 | 0.9941 | 0.9798 | 0.9726 | |
| 1462 | 567 | 1300 | 352 | 439 | 341 | 792 | 620 | |
| 0.0365 | 0.0591 | 0.0383 | 0.0714 | 0.0670 | 0.0762 | 0.0495 | 0.0556 | |
| 0.0036 | 0.0035 | 0.0047 | 0.0137 | 0.0045 | 0.0041 | 0.0050 | 0.0066 | |
| 0.9813 | 0.9954 | 0.9745 | 0.9447 | 0.9914 | 0.9962 | 0.9789 | 0.9699 | |
| 1233 | 433 | 1100 | 253 | 348 | 261 | 662 | 531 | |
| 0.0397 | 0.0677 | 0.0418 | 0.0852 | 0.0753 | 0.0873 | 0.0541 | 0.0600 | |
| 0.0039 | 0.0033 | 0.0047 | 0.0144 | 0.0050 | 0.0038 | 0.0056 | 0.0074 | |
| 0.9797 | 1.0000 | 0.9747 | 0.9494 | 0.9887 | 1.0000 | 0.9782 | 0.9712 | |
| 1033 | 327 | 947 | 178 | 265 | 197 | 550 | 451 | |
| 0.0433 | 0.0782 | 0.0451 | 0.1020 | 0.0861 | 0.1008 | 0.0593 | 0.0652 | |
| 0.0044 | 0.0000 | 0.0051 | 0.0164 | 0.0065 | 0.0000 | 0.0062 | 0.0079 | |
| 0.9828 | 1.0000 | 0.9740 | 0.9752 | 0.9898 | 1.0000 | 0.9824 | 0.9792 | |
| 815 | 244 | 807 | 121 | 197 | 160 | 454 | 385 | |
| 0.0489 | 0.0905 | 0.0488 | 0.1262 | 0.1000 | 0.1118 | 0.0655 | 0.0710 | |
| 0.0046 | 0.0000 | 0.0056 | 0.0141 | 0.0071 | 0.0000 | 0.0062 | 0.0073 | |
| 0.9814 | 1.0000 | 0.9729 | 0.9762 | 0.9874 | 1.0000 | 0.9790 | 0.9785 | |
| 646 | 183 | 663 | 84 | 159 | 116 | 381 | 325 | |
| 0.0549 | 0.1045 | 0.0538 | 0.1515 | 0.1111 | 0.1313 | 0.0713 | 0.0772 | |
| 0.0053 | 0.0000 | 0.0063 | 0.0166 | 0.0088 | 0.0000 | 0.0073 | 0.0081 | |
| 0.9828 | 1.0000 | 0.9801 | 0.9825 | 1.0000 | 1.0000 | 0.9834 | 0.9752 | |
| 524 | 134 | 554 | 57 | 111 | 86 | 301 | 282 | |
| 0.0610 | 0.1222 | 0.0592 | 0.1849 | 0.1342 | 0.1525 | 0.0805 | 0.0826 | |
| 0.0057 | 0.0000 | 0.0059 | 0.0174 | 0.0000 | 0.0000 | 0.0074 | 0.0093 | |

b. Example text output for SLE_threshold.C

A data block was generated for each middle scintillator and then those values were inserted into the program 'Surface.C'

The top row contains the number of trigger events for each value of *threshold_a* (increasing)

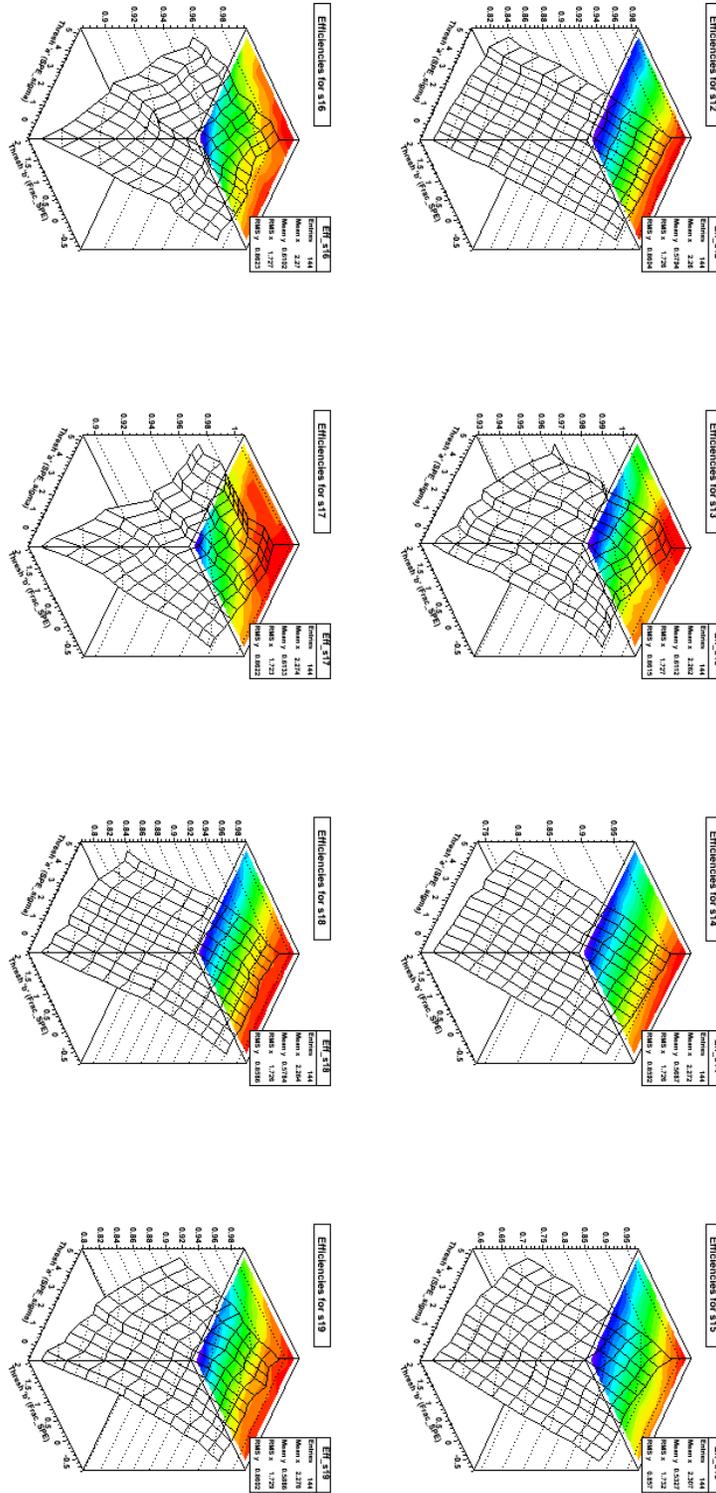
Threshold_a increases down the columns

Threshold_b increases across the rows

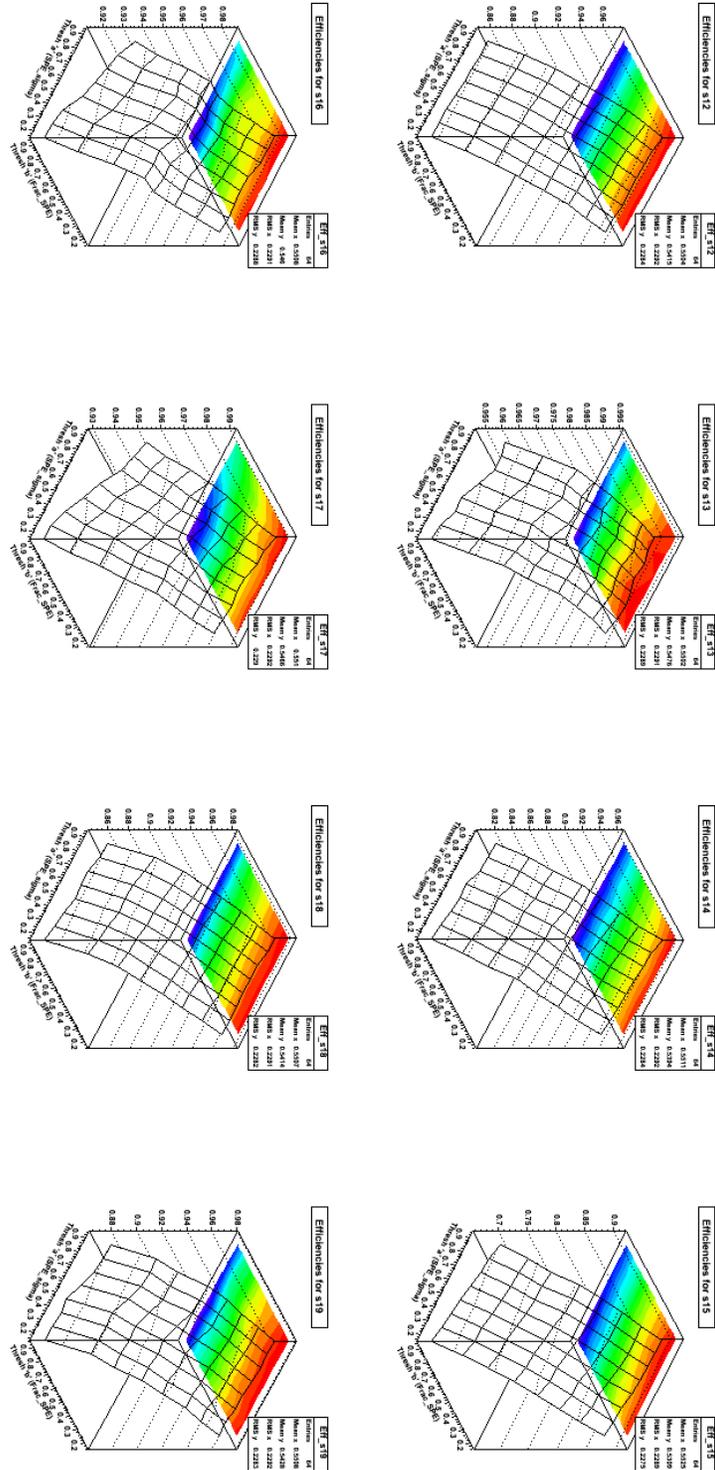
For s12:

```
2786 2629 2445 2311 2172 1991 1798 1627 1462 1302 1159 1014
0.9620, 0.9526, 0.9419, 0.9286, 0.9139, 0.9009, 0.8848, 0.8693, 0.8579, 0.8388, 0.8234, 0.8087,
0.9623, 0.9536, 0.9429, 0.9296, 0.9156, 0.9030, 0.8859, 0.8703, 0.8585, 0.8406, 0.8262, 0.8113,
0.9616, 0.9538, 0.9431, 0.9301, 0.9170, 0.9043, 0.8871, 0.8716, 0.8593, 0.8409, 0.8258, 0.8110,
0.9641, 0.9563, 0.9455, 0.9325, 0.9191, 0.9061, 0.8884, 0.8728, 0.8602, 0.8412, 0.8265, 0.8118,
0.9655, 0.9581, 0.9466, 0.9342, 0.9222, 0.9093, 0.8909, 0.8762, 0.8633, 0.8439, 0.8283, 0.8131,
0.9684, 0.9608, 0.9493, 0.9377, 0.9262, 0.9126, 0.8940, 0.8800, 0.8669, 0.8483, 0.8322, 0.8167,
0.9683, 0.9600, 0.9494, 0.9372, 0.9260, 0.9132, 0.8943, 0.8810, 0.8693, 0.8498, 0.8331, 0.8176,
0.9668, 0.9582, 0.9471, 0.9348, 0.9244, 0.9121, 0.8949, 0.8808, 0.8697, 0.8494, 0.8322, 0.8187,
0.9692, 0.9610, 0.9508, 0.9384, 0.9282, 0.9159, 0.8981, 0.8844, 0.8735, 0.8543, 0.8372, 0.8235,
0.9693, 0.9601, 0.9493, 0.9363, 0.9270, 0.9140, 0.8986, 0.8856, 0.8740, 0.8541, 0.8379, 0.8257,
0.9681, 0.9586, 0.9474, 0.9336, 0.9232, 0.9120, 0.8965, 0.8818, 0.8697, 0.8499, 0.8352, 0.8240,
0.9704, 0.9625, 0.9527, 0.9389, 0.9300, 0.9201, 0.9063, 0.8974, 0.8856, 0.8659, 0.8511, 0.8422,
```

c. Graphical Output for SLE_thresholds_ab.C for first chart of threshold values



d. Graphical Output for SLE_thresholds_ab.C for second chart of threshold values



TRENDS AND CONCLUSIONS

4.1 Evidence for Leakage

4.1.1 Understanding the Trend

Focusing on the PMT layout diagrams of Section 3.1.2 a-d, we see that neighboring channels on the PMT (those with a 'TypeI' relation) demonstrate pronounced light sharing. Other channels on the same PMT demonstrate minimal light sharing that is greater than, but of the same magnitude as, the 'above threshold response' on the other PMT. This opposite PMT correspondence cannot physically be due to 'leakage' at the face of the PMT. From the histograms and data of Section 3.1.2 e-f, we know that we should expect 'leakage' on the order of 1PE in each channel with a TypeI relation to a scintillator with a through-going cosmic ray with a probability of ~7-8%. Interestingly, for TypeII/III relations and especially Type IV relations, the order of magnitude for 'leakage' is less 'focused' around 1PE and is skewed high. This skewing is probably a factor of viewing all of the scintillators for a given type on the same histogram. Because we are viewing seven scintillators for TypeII/III and 14 scintillators for TypeIV, we are essentially magnifying pedestal noise (or noise from elsewhere) for these channels.

4.1.2 'Left' vs 'Right' side of Apparatus

The left and right side of the apparatus, corresponding to the right and left PMTs (respectively), demonstrate similar trends, intensities and probabilities for 'leakage.' Neither PMT dominates. For example, from Section 3.1.2 e, we see that while the right PMT has a higher expectation for TypeI leakage by ~15%, the opposite is true for TypeII/III and Type IV relations.

4.2 Leakage and Efficiency

4.2.1 Understanding the Trend

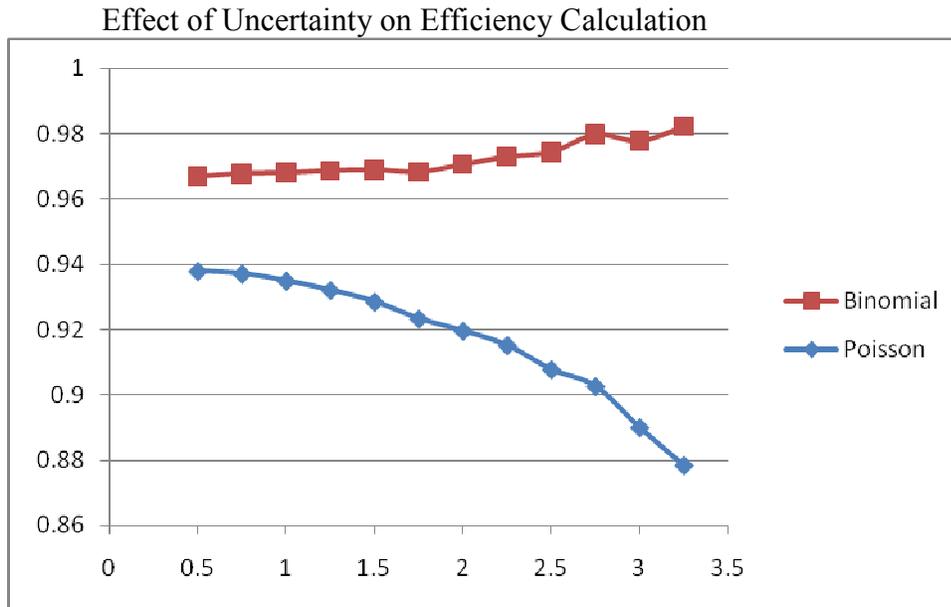
There are two potential effects that 'Leakage' at the face of the PMT could have when calculating SLE. If the 'above threshold response' in the trigger scintillators is due to leakage rather than a through-going cosmic ray, the denominator will have too many events and the efficiency will thus be biased low. If the 'above threshold response' in the middle scintillators is due to leakage rather than a through-going cosmic ray, the numerator will be too large and thus the efficiency will be biased high.

We first increase *threshold_a* in order to increase the purity of the events in the denominator. Increasing this threshold increases purity because the probability of having an above threshold response due to pedestal noise or 'leakage' decreases as we increase nPE. This is only compounded by the fact that we have two triggers. Also, because cosmic ray muons are a 'minimum ionizing particle,' the nPE excited by the through-going cosmic ray depends only on distance travelled through the scintillating material.²¹ In other words, the mean for what should be a Poisson distribution of above threshold response in the middle scintillators should not change as we adjust the trigger thresholds. Therefore, increasing trigger thresholds should increase purity without altering other factors. However, the price of increasing purity is

²¹ Amsler 27.5

decreasing the number of events that meet the condition. This thereby increases the statistical error and such increases in error can effectively nullify any gains in calculated efficiency. The effect is plotted below and the different methods of calculating these errors are discussed in Section 4.2.2. The data below²² represents (SLE – ‘1σ’ of the associated error) averaged over s12-19 and the x-axis is the change in ‘a’ for:

$$\text{threshold}_a = a \times \mu_{SPE}$$



So while the surface plots in Section 3.2.2 demonstrate that increasing the trigger threshold yields an increase in efficiency, the increase is counterproductive (at least using Poisson errors) as the uncertainty in the efficiency increases faster than the efficiency.

Thus the introduction of the novel scale used for both thresholds²³ in Section 3.2.2 *d*: By using these specific values for ‘a’ and ‘b,’ we increase the thresholds in such a way that the area under the SPE peak increases linearly as we increase the thresholds. This condition is secured by using the cumulative distribution function (*cdf*) for a Gaussian²⁴ where:

$$cdf = \frac{1}{2} \operatorname{erf}\left(\frac{z}{\sqrt{2}}\right) + \frac{1}{2}$$

$$\operatorname{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt$$

The following were solved numerically using the online graphing program ‘Wolfram Alpha’

| | | | | | | | | |
|--------|--------|--------|--------|-----|-------|-------|-------|-------|
| cdf(z) | 0.2 | 0.3 | 0.4 | 0.5 | 0.6 | 0.7 | 0.8 | 0.8 |
| z | -0.842 | -0.524 | -0.253 | 0.0 | 0.253 | 0.524 | 0.842 | 1.282 |

²² Output from Efficiency_a.C manipulated with Excel

²³ And of the programs that use these thresholds: Efficiency_ab.C and Surface.C

²⁴ Wolfram

Recall that the definition of the z standard variable is:

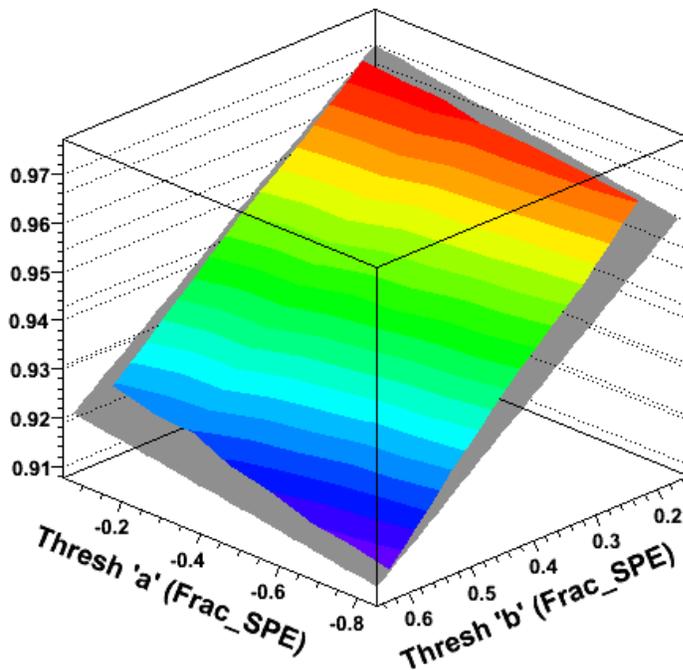
$$z \equiv \frac{x - \mu}{\sigma} \quad \text{which rearranged is:} \quad x = \mu + z \times \sigma = \text{threshold_}a/b$$

If the effect of ‘leakage’ on efficiency is linear and if ‘leakage’ can be accurately modeled by a Gaussian, then changing the thresholds in the manner described above will change the total fraction of leakage at a constant rate which should then affect efficiency linearly. This is done in Section 3.2.2 d for each middle scintillator. This is appealing because it allows us to make predictions for the effect of leakage on SLE for large values of sigma using data points in a narrow range.

We see that as we increase *threshold_a* in this manner, we get a visually ‘nice’ linear increase in efficiency. However, as we increase *threshold_b*, we at first observe what appears to be a negative linear response in calculated SLE. However, as *threshold_b* approaches 1.0, the decrease in efficiency increases more rapidly. This is at least partly accounted for by the increasing number of actual cosmic ray events that will not be counted.²⁵

By averaging the efficiencies for the middle scintillators, inverting the scaling for *threshold_a*²⁶ and by only considering $z = 0.2$ to $z = 0.6$ for *threshold_b*, we can develop a linear model for the effect of leakage on SLE. Furthermore, the constant in this particular model corresponds to a theoretical SLE with zero leakage.

$$\boxed{[0]+[1]*x+[2]*y}$$



Parameters drawn from averaged efficiencies

0.988207 w/ error 0.5517
is the constant and
THEORETICAL SLE

0.0130706 w/ error 0.6692
is the change in
efficiency per fraction
SPE of the trigger
scintillators

-0.103619 w/ error 1.084
is the change in
efficiency per fraction
SPE of the middle
scintillators

²⁵ Recall that even if the triggers are well above threshold, the middle scintillators should remain Poisson distributed so that there will be a portion of what had been ‘above threshold response’ below the new cut

²⁶ $\text{threshold_}a \rightarrow -1.0*(1.0-\text{threshold_}a)$

4.2.2 Statistical Uncertainty

There are two common methods for calculating the statistical uncertainty for efficiency. The first uses the formula for the propagation of errors and assumes the characteristics of a Poisson distribution and the second considers the process of ‘applying thresholds when determining efficiency’ to be a binomial process. The Binomial method is best suited for calculating the errors associated with efficiencies that are neither near 0.0 nor 1.0 and since these efficiencies are very near to 1.0. Therefore, where not otherwise specified, we have used the larger Poisson uncertainties in the discussion of data trends.²⁷

Poisson:

$$\delta\varepsilon = \varepsilon \sqrt{\left(\frac{\delta k}{k}\right)^2 + \left(\frac{\delta N}{N}\right)^2} = \frac{k}{N} \sqrt{\frac{1}{k} + \frac{1}{N}}$$

Binomial:

$$\delta\varepsilon = \frac{\sigma_k}{N} = \frac{1}{N} \sqrt{\varepsilon(1-\varepsilon)N} = \frac{1}{N} \sqrt{k\left(1 - \frac{k}{N}\right)}$$

4.3 Recommendations for Future Study

Much of the analysis in this note (and the analysis in the several others that I have referenced) relies on the fact that the excess seen above what should be a Poisson distribution for each scintillator is taken to be the single photoelectron peak. The PMTs could be tested and calibrated with a known intensity light source to verify this assumption and to provide confidence to these results and others. It seems possible that as an alternative to the SPE, the excess is in fact the direct result of ‘leakage.’ It is plausible to imagine adding the characteristic shapes for Type I leakage to Poisson distributions to achieve the shape of the histograms for each scintillator. If the excess is indeed due specifically to leakage, it could still be of order 1PE, but it would not have to be so *a priori*.

²⁷ Paterno, 3.

REFERENCES AND APPENDICES

5.1 Works Cited

Amsler *et al.*. “Passage of Particles through Matter.” PL **B667**, 1 (2008) and 2009 partial update for the 2010 edition (<http://pdg.lbl.gov>).

“CDF for Normal Distribution.” Wolfram Alpha. 8 May 2011.
<<http://www.wolframalpha.com/input/?i=cdf+for+normal+distribution>>

Glenzinski *et al.*. “Results from the CRV Prototype at Fermilab using Summer 2009 Data.” Mu2e-doc-777. Fermilab: The Mu2e Collaboration, 2010.

Glenzinski *et al.*. “Results from the CRV Prototype at Fermilab using Summer 2010 Data.” Mu2e-doc-1382. Fermilab: The Mu2e Collaboration, 2011.

Marciano, Willaim J., Toshironori Mori, and J. Michael Roney. “Charged Lepton Flavor Violation Experiments.” *The Annual Review of Nuclear and Particle Science*. 58 (2008): 315-341.

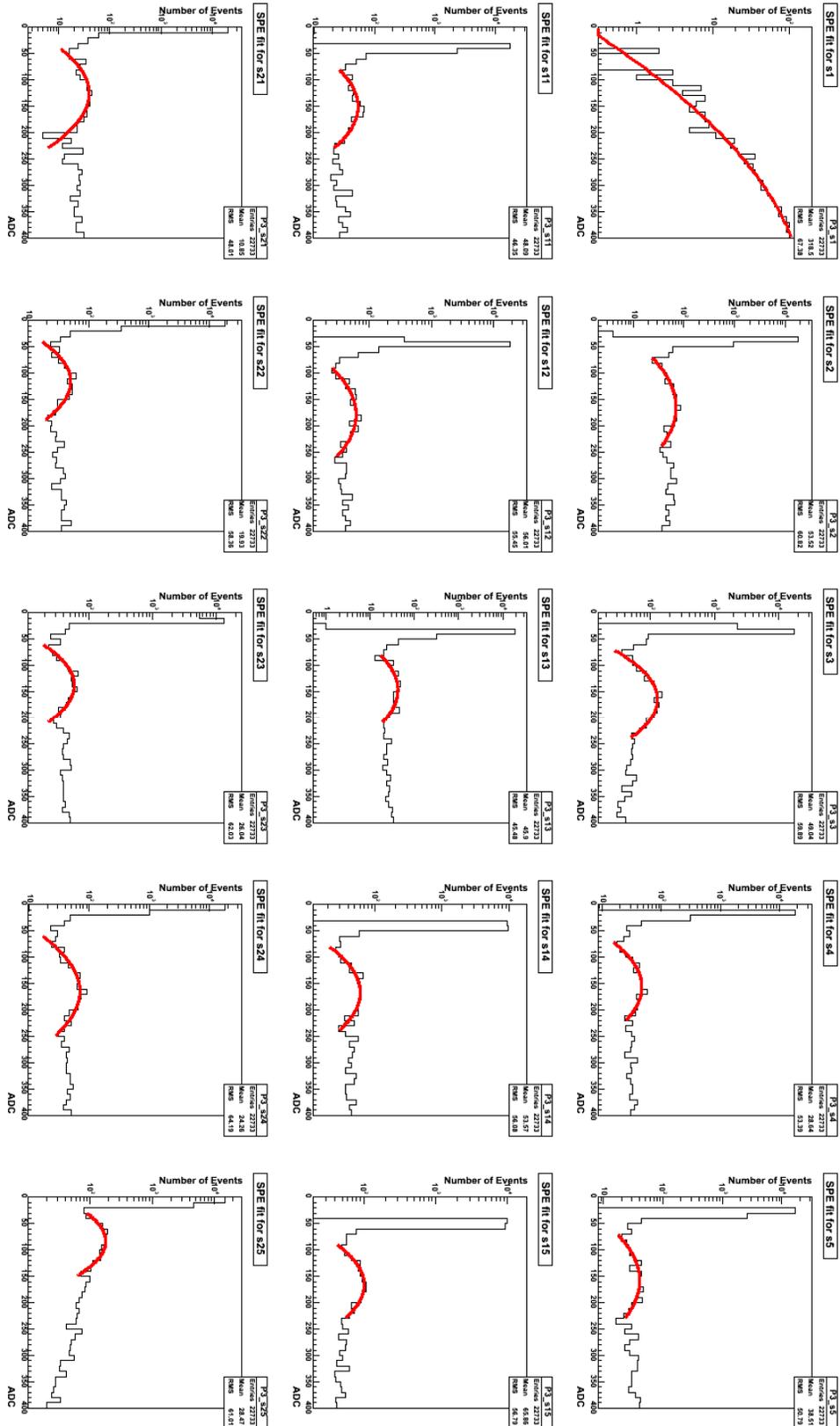
Hyperphysics. Georgia State University. 8 May 2011.
<<http://hyperphysics.phy-astr.gsu.edu/hbase/hph.html>>.

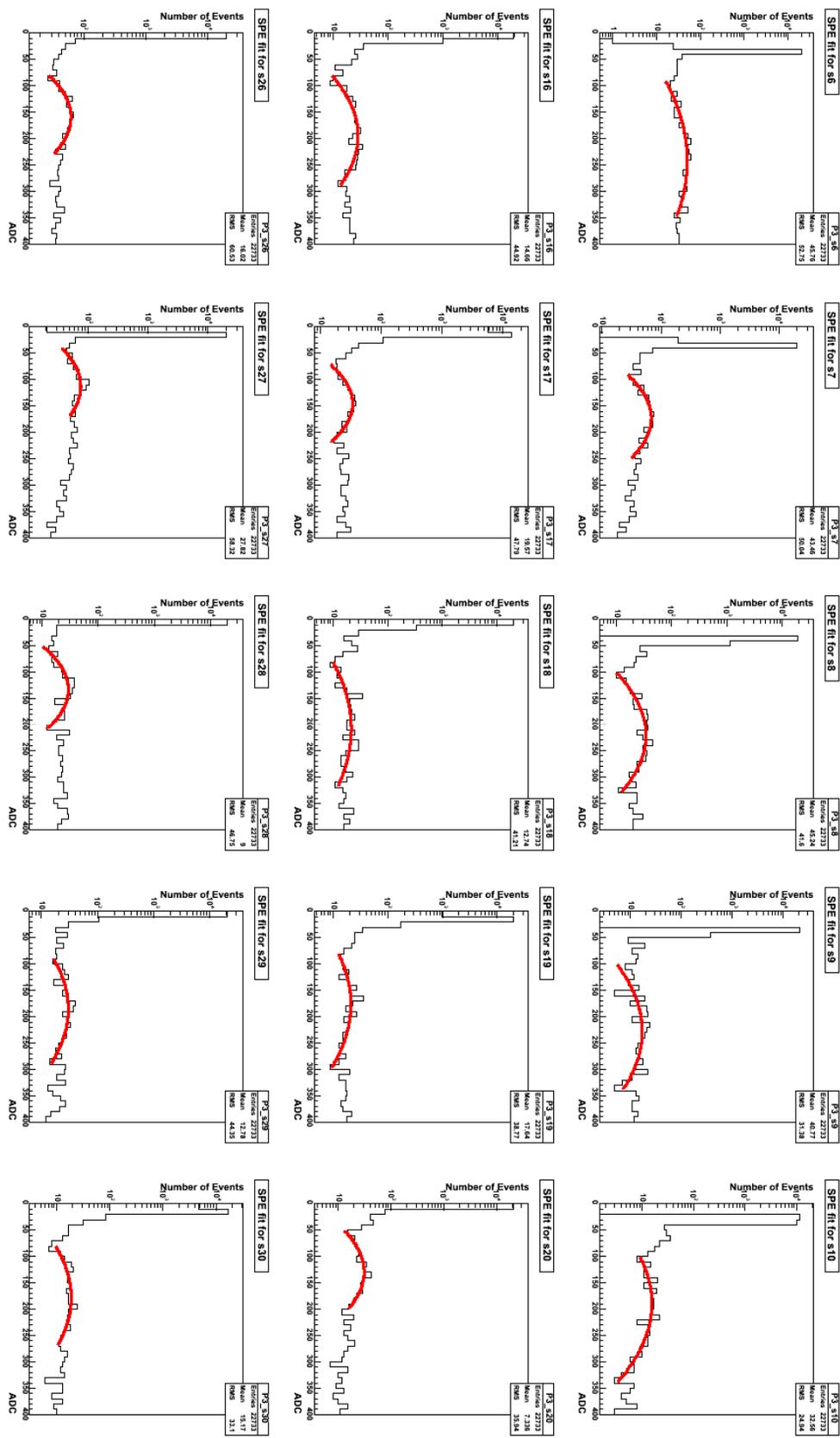
Paterno, Marc. “Calculating Efficiencies and Their Uncertainties.” FNAL, 2003.

“Proposal to Search for $\mu^- N \rightarrow e^- N$ with a Single Event Sensitivity Below 10^{-16} .” Fermilab: The Mu2e Collaboration, 2008.

S. I. Collaboration, *Eur. Phys. J. C* 47, 337 (2006).

6.2 SPE Peaks





5.3 Data Processing Algorithms

5.3.1 Evidence for Leakage

PMT_chart.C

```
{
//Set the scintillators that you desire to test against
//(i.e. require this scintillator to be above threshold and then see occupancy of others but in
PMT schematic order
int test1 = 5;
int test2 = 15;
int test3 = 25;

//int test1 = 6;
//int test2 = 16;
//int test3 = 26

gROOT->ForceStyle();
gROOT->SetStyle("Plain");
gStyle->SetPalette(1);

//Determined by inspection of the minimum for P3
float thresholds[32] = {0.0, 0.0, 70.0, 70.0, 70.0, 70.0, 90.0, 90.0, 100.0, 100.0, 100.0,
                        80.0, 90.0, 80.0, 80.0, 90.0, 80.0, 70.0, 80.0, 80.0, 50.0,
                        40.0, 40.0, 60.0, 60.0, 30.0, 80.0, 40.0, 50.0, 90.0, 80.0, 0.0};

//SPE data from SPE_P3.C
float SPE_mean[32] = {0.0, 535.54, 164.95, 164.60, 156.23, 158.08, 242.23, 173.04, 220.08,
227.65, 189.09,
                        150.40, 178.75, 146.53, 166.79, 165.97, 194.37, 145.65, 208.20, 176.26,
129.30,
                        125.46, 116.04, 137.64, 164.51, 83.40, 159.28, 113.11, 132.74, 185.54,
178.19, 0.0};

float SPE_sigma[32] = {0.0, 147.01, 64.25, 53.30, 56.86, 66.92, 100.89, 60.87, 76.10, 84.19,
85.30,
                        58.49, 66.30, 49.03, 59.69, 57.57, 77.03, 57.17, 102.89, 94.51, 60.08,
53.61, 51.55, 51.64, 61.61, 45.14, 61.86, 61.10, 56.97, 86.76, 83.53,
0.0};

//loop over the file
for (int ifile=3; ifile<=3; ifile++) {

//Get the file (in quotes with '.root')
string infile = Form("%d.root",ifile);
TFile *_file0 = TFile::Open(infile.c_str());

//Get the tree and the leaves
tv__tree = (TTree *) gROOT->FindObject("ntuple");
TLeaf *l_s[32];
l_s[0] = tv__tree->FindLeaf("bottomtrig");
l_s[31] = tv__tree->FindLeaf("toptrig");
for ( int ileaf = 1; ileaf <= 30; ileaf++ ) {
    string s_leaf = Form("s%d",ileaf);
    l_s[ileaf] = tv__tree->FindLeaf(s_leaf.c_str());
}

//Get the number of events
int nevent = int(tv__tree->GetEntries());
cout<<nevent<<endl<<endl;

// hot[event#]==1 iff the event is hot (i.e. greater than than 50percent of scint. above
threshold)
int hot[nevent];
int lit=0;
int hot_total=0;
```

```

for ( int iloop = 0; iloop < nevent; iloop++) {
    tv__tree->GetEntry(iloop);

    lit = 0;
    for (int ileaf=1; ileaf<=30; ileaf++) {
        if ( l_s[ileaf]->GetValue() > thresholds[ileaf]) {
            lit++;
        }
    }

    if (lit>15) {
        hot_total++;
        hot[iloop]=1;
    }
}
cout<<nevent<<endl;
cout<<hot_total<<endl;
float percent_hot = float(hot_total)/float(nevent);
cout<<percent_hot<<endl;
cout<<(nevent - hot_total)<<endl;

//Book histograms
TH1F *hist_array[31];
int bins = 150;
for (int ihist=1; ihist<=30; ihist++) {
    string hist_ADC = Form("P3_s%dADC",ihist);
    string nameADC = Form("s%d given s%d, s%d, and s%d 'lit'",ihist,test1,test2,test3);
    hist_array[ihist] = new TH1F(hist_ADC.c_str(),nameADC.c_str(),bins,0.0,1500.0);
}

//Loop over the events
for (int ievent=0; ievent<nevent; ievent++) {

    //Exclude hot events
    if (hot[ievent] != 1) {

        //Get entries for each event
        tv__tree->GetEntry(ievent);

        for (int ihist=1; ihist<=30; ihist++){

            float cond1 = float(l_s[test1]->GetValue());
            float cond2 = float(l_s[test2]->GetValue());
            float cond3 = float(l_s[test3]->GetValue());

            if((cond1>thresholds[test1]) && (cond2>thresholds[test2]) && (cond3>thresholds[test3])
) {

                float x = float(l_s[ihist]->GetValue());
                hist_array[ihist].Fill(x);
            }
        }
    }
}

//Print histograms
TCanvas *c_1 = new TCanvas("c_1",infile.c_str(),0,0,1600,1600);
c_1->Divide(4,4);

//Left PMT
/*
for (int ipad=2; ipad<=16; ipad++) {

    c_1->cd(ipad);

    if (ipad==2) hist_array[6]->Draw();
    if (ipad==3) hist_array[7]->Draw();
    if (ipad==4) hist_array[8]->Draw();
    if (ipad==5) hist_array[9]->Draw();
}
*/

```

```

    if (ipad==6) hist_array[10]->Draw();
    if (ipad==7) hist_array[16]->Draw();
    if (ipad==8) hist_array[17]->Draw();
    if (ipad==9) hist_array[18]->Draw();
    if (ipad==10) hist_array[19]->Draw();
    if (ipad==11) hist_array[20]->Draw();
    if (ipad==12) hist_array[26]->Draw();
    if (ipad==13) hist_array[27]->Draw();
    if (ipad==14) hist_array[28]->Draw();
    if (ipad==15) hist_array[29]->Draw();
    if (ipad==16) hist_array[30]->Draw();

    gPad->SetLogy();
}
*/

//Right PMT
for (int ipad=1; ipad<=16; ipad++) {

    c_1->cd(ipad);

    if (ipad==1) hist_array[3]->Draw();
    if (ipad==2) hist_array[14]->Draw();
    if (ipad==3) hist_array[25]->Draw();
    if (ipad==4) hist_array[21]->Draw();
    if (ipad==5) hist_array[4]->Draw();
    if (ipad==6) hist_array[15]->Draw();
    if (ipad==7) hist_array[11]->Draw();
    if (ipad==8) hist_array[22]->Draw();
    if (ipad==9) hist_array[5]->Draw();
    if (ipad==10) hist_array[1]->Draw();
    if (ipad==11) hist_array[12]->Draw();
    if (ipad==12) hist_array[23]->Draw();
    //if (ipad==13) hist_array[27]->Draw();
    if (ipad==14) hist_array[2]->Draw();
    if (ipad==15) hist_array[13]->Draw();
    if (ipad==16) hist_array[24]->Draw();

    if (ipad != 13) gPad->SetLogy();
}
    c_1.Print("PMT_right.gif");
}
}

```

Leakage_LR.C

```

{
//Input:
// Program Runs P3
// 2 halves: Left and Right PMT

//Program:
// Catalogs above threshold response given s5, s15, s25 or s6, s16, s26
// Correlations using 2D histograms

//Output:
// Avg leakage in
// plots of 2D histograms with linear fit for left and right PMT

gROOT->ForceStyle();
gROOT->SetStyle("Plain");
gStyle->SetPalette(1);

//Backup to file
streambuf *output_buf;
ofstream output_stream;
output_stream.open("Leakage_LR.txt");
output_buf = output_stream.rdbuf();

```

```

cout.rdbuf(output_buf);

//Determined by inspection of the minimum for P3
float thresholds[32] = {0.0, 0.0, 70.0, 70.0, 70.0, 70.0, 90.0, 90.0, 100.0, 100.0, 100.0,
                      80.0, 90.0, 80.0, 80.0, 90.0, 80.0, 70.0, 80.0, 80.0, 50.0,
                      40.0, 40.0, 60.0, 60.0, 30.0, 80.0, 40.0, 50.0, 90.0, 80.0, 0.0};

//SPE data from SPE_P3.C
float SPE_mean[32]={0.0,535.54,164.95,164.60,156.23,158.08,242.23,173.04,220.08,227.65,189.09,
                  150.40,178.75,146.53,166.79,165.97,194.37,145.65,208.20,176.26,129.30,
                  125.46,116.04,137.64,164.51,83.40,159.28,113.11,132.74,185.54,178.19,0.0};

float SPE_sigma[32]={0.0,147.01,64.25,53.30,56.86,66.92,100.89,60.87,76.10,84.19,85.30,
                   58.49,66.30,49.03,59.69,57.57,77.03,57.17,102.89,94.51,60.08,
                   53.61,51.55,51.64,61.61,45.14,61.86,61.10,56.97,86.76,83.53,0.0};

// Loop over the file
for (int ifile=3; ifile<=3; ifile++) {

    //Get the file (in quotes with '.root')
    string infile = Form("P%d.root",ifile);
    TFile *_file0 = TFile::Open(infile.c_str());

    //Get the tree and the leaves
    tv__tree = (TTree *) gROOT->FindObject("ntuple");
    TLeaf *l_s[32];
    l_s[0] = tv__tree->FindLeaf("bottomtrig");
    l_s[31] = tv__tree->FindLeaf("toptrig");
    for ( int ileaf = 1; ileaf <= 30; ileaf++ ) {
        string s_leaf = Form("s%d",ileaf);
        l_s[ileaf] = tv__tree->FindLeaf(s_leaf.c_str());
    }

    //Get the number of events
    int nevent = int(tv__tree->GetEntries());
    cout<<nevent<<" events in P"<<ifile<<endl<<endl;

    // hot[event#]==1 iff the event is hot (i.e. greater than than 50percent of scint. above
    threshold)
    // only okay because nevent for P1 is greatest...Why can't we redim with each file?
    int hot[nevent];
    int lit=0;
    int hot_total=0;

    for ( int iloop = 0; iloop < nevent; iloop++ ) {
        tv__tree->GetEntry(iloop);

        lit = 0;
        for (int ileaf=1; ileaf<=30; ileaf++) {
            if ( l_s[ileaf]->GetValue() > thresholds[ileaf]) {
                lit++;
            }
        }

        if (lit>15) {
            hot_total++;
            hot[iloop]=1;
        }
    }

    cout<<(nevent - hot_total)<<" events not 'hot'"<<endl<<endl;
    nevent = nevent - hot_total;

    //Book histograms
    TH2F *hist_array[2];
    int bins = 20;
    string hist_right = Form("P3_right_PMT");
    string hist_left = Form("P3_left_PMT");
    string name_right = Form("Type I events s5,15,25");

```

```

string name_left = Form("Type I events s6,16,26");
hist_array[0] = new TH2F(hist_right.c_str(),name_right.c_str(),bins,0.0,20.0,bins,0.0,5.0);
hist_array[1] = new TH2F(hist_left.c_str(),name_left.c_str(),bins,0.0,20.0,bins,0.0,5.0);
for (int ihist=0; ihist<=1; ihist++) {
    hist_array[ihist]->GetXaxis()->SetTitle("AVG nPE for Triggers");
    hist_array[ihist]->GetXaxis()->SetLabelSize(.03);
    hist_array[ihist]->GetYaxis()->SetTitle("nPE Type I 'Leakage'");
    hist_array[ihist]->GetYaxis()->SetLabelSize(.03);
}

//Set cells of interest
int trig[2][3] = {5,15,25,
                  6,16,26};
int trig_n[2] = {0,0};

//s1 and s10 left out
int typeI[2][4] = {4,11,14,21,
                  7,17,20,30};
int typeI_n[2] = {0,0};

int typeIII[2][7] = {2,3,12,13,22,23,24,
                    8,9,18,19,27,28,29};
int typeIII_n[2] = {0,0};

//s1 and s10 left out
int typeIV[2][14] = {6,7,8,9,16,17,18,19,20,26,27,28,29,30,
                    2,3,4,5,11,12,13,14,15,21,22,23,24,25};
int typeIV_n[2] = {0,0};

// Loop over each PMT
for (int iside=0; iside<=1; iside++) {

    //Loop over the events
    for (int ievent=0; ievent<nevent; ievent++) {

        //Exclude hot events
        if (hot[ievent] != 1) {

            //Get entries for each event
            tv_tree->GetEntry(ievent);

            //Did the cosmic ray go through s5,s15,s25 or s6,s16,s26
            int ray = 0;
            float xfill = 0.0;
            for (int itrig=0; itrig<3; itrig++) {
                int scint = trig[iside][itrig];
                float entry = float( l_s[scint]->GetValue() );
                float cond = float( thresholds[scint] );
                if (entry > cond) {
                    ray++;
                    xfill = xfill + (entry / SPE_mean[scint]);
                }
            }

            //Fill the histograms if the cosmic ray was above threshold
            if (ray == 3) {
                trig_n[iside] = trig_n[iside] + 1;
                xfill = xfill / 3.0;

                // Type I
                for (int il=0; il<4; il++) {
                    scint = typeI[iside][il];
                    entry = float( l_s[scint]->GetValue() );
                    cond = float( thresholds[scint] );
                    if (entry > cond) {
                        typeI_n[iside] = typeI_n[iside] + 1;
                        float yfill = entry / SPE_mean[scint];
                    }
                }
            }
        }
    }
}

```

```

        hist_array[iside].Fill(xfill,yfill);
    }
}

// Type II/III
for (int i3=0; i3<7; i3++) {
    scint = typeIII[iside][i3];
    entry = float( l_s[scint]->GetValue() );
    cond = float( thresholds[scint] );
    if (entry > cond) {
        typeIII_n[iside] = typeIII_n[iside] + 1;
    }
}

// Type IV
for (int i4=0; i4<14; i4++) {
    scint = typeIV[iside][i4];
    entry = float( l_s[scint]->GetValue() );
    cond = float( thresholds[scint] );
    if (entry > cond) {
        typeIV_n[iside] = typeIV_n[iside] + 1;
    }
}
}
}
}

// Print results
for (int iside=0; iside<=1; iside++){

    float trig_percent = float(trig_n[iside]) / float(nevent);
    cout<<trig_percent<<" percent of events for PMT "<<iside<<" with all three trigger scint
above threshold"<<endl;
    cout<<trig_n[iside]<<" number of such events"<<endl<<endl;

    float typeI_prob = float(typeI_n[iside]) / (4.0 * trig_n[iside]);
    cout<<typeI_prob<<" TypeI leakage exp prob for PMT "<<iside<<endl;
    cout<<typeI_n[iside]<<" number of such events (4 possible scint per 'event')"<<endl<<endl;

    float typeIII_prob = float(typeIII_n[iside]) / (7.0 * trig_n[iside]);
    cout<<typeIII_prob<<" TypeII/III leakage exp prob for PMT "<<iside<<endl;
    cout<<typeIII_n[iside]<<" number of such events (7 possible scint per
'event')"<<endl<<endl;

    float typeIV_prob = float(typeIV_n[iside]) / (14.0 * trig_n[iside]);
    cout<<typeIV_prob<<" TypeIV leakage exp prob for PMT "<<iside<<endl;
    cout<<typeIV_n[iside]<<" number of such events (14 possible scint per
'event')"<<endl<<endl;

    cout<<endl;
}

//Print histograms
TCanvas *c_1 = new TCanvas("c_1",infile.c_str(),0,0,800,400);
c_1->Divide(2);
for (int ipad=1; ipad<=2; ipad++) {
    c_1->cd(ipad);
    hist_array[ipad-1]->Draw("BOX");
}

c_1.Print("Leakage_LR.gif");
}
}
}

```

5.3.2 Leakage and Efficiency

Efficiency_a.C

```
#include <iostream>
#include <iomanip>

int Efficiency_a()
{
    //Backup to file
    stringstream *output_buf;
    ofstream output_stream;
    output_stream.open("Efficiency_a.txt");
    output_buf = output_stream.rdbuf();
    cout.rdbuf(output_buf);

    //Input:
    // get 'thresholds[]' and 'SPE_mean[]/sigma[]' from 'SPE_P3.C'

    //Program:
    // loops over P1 through P7 calculating single channel efficiency of scintillators 12-19
    // the effect of overlap is taken into consideration with an 'or' statement

    gROOT->ForceStyle();
    gROOT->SetStyle("Plain");

    //Determined by inspection of the minimum for P3
    float thresholds[32] = {0.0, 0.0, 70.0, 70.0, 70.0, 70.0, 90.0, 90.0, 100.0, 100.0, 100.0,
                          80.0, 90.0, 80.0, 80.0, 90.0, 80.0, 70.0, 80.0, 80.0, 50.0,
                          40.0, 40.0, 60.0, 60.0, 30.0, 80.0, 40.0, 50.0, 90.0, 80.0, 0.0};

    //SPE data from SPE_P3.C
    float SPE_mean[32] = {0.0, 535.54, 164.95, 164.60, 156.23, 158.08, 242.23, 173.04, 220.08, 227.65, 189.09,
                        150.40, 178.75, 146.53, 166.79, 165.97, 194.37, 145.65, 208.20, 176.26, 129.30,
                        125.46, 116.04, 137.64, 164.51, 83.40, 159.28, 113.11, 132.74, 185.54, 178.19, 0.0};

    float SPE_sigma[32] = {0.0, 147.01, 64.25, 53.30, 56.86, 66.92, 100.89, 60.87, 76.10, 84.19, 85.30,
                          58.49, 66.30, 49.03, 59.69, 57.57, 77.03, 57.17, 102.89, 94.51, 60.08,
                          53.61, 51.55, 51.64, 61.61, 45.14, 61.86, 61.10, 56.97, 86.76, 83.53, 0.0};

    gStyle->SetPalette(1);

    //Allocate array indicating whether or not an event is 'hot' outside the loop as a pointer
    int* hot = NULL;

    //Loop over P1.root through P7.root
    for (int ifile=3; ifile<=3; ifile++){

        string infile = Form("P%d.root",ifile);
        TFile *_file0 = TFile::Open(infile.c_str());

        cout<<endl<<endl<<"For "<<infile<<": "<<endl;

        //Get the tree
        tv__tree = (TTree *) gROOT->FindObject("ntuple");

        //Get the leaves
        TLeaf *l_s[32];
        l_s[0] = tv__tree->FindLeaf("bottomtrig");
        l_s[31] = tv__tree->FindLeaf("toptrig");
        for ( int ileaf = 1; ileaf <= 30; ileaf++ ) {
            string s_leaf =Form("s%d",ileaf);
            l_s[ileaf] = tv__tree->FindLeaf(s_leaf.c_str());
        }

        // 'nevent' is the number of events
        Int_t nevent = int(tv__tree->GetEntries());
    }
}
```

```

//*****
// 'hot[event#]==1' iff the event is hot (i.e. greater than than 50percent of scint. above
threshold)
// requires some squirrely programming due to inability to redim an array in C++
delete [] hot;
hot = NULL;
hot = new int[nevent];
for (int ihot=0; ihot<nevent; ihot++) {
    hot[ihot] = 0;
}
int lit=0;
int hot_total=0;

for ( int iloop = 0; iloop < nevent; iloop++) {
    tv_tree->GetEntry(iloop);

    lit = 0;
    for (int ileaf=1; ileaf<=30; ileaf++) {
        if ( l_s[ileaf]->GetValue() > thresholds[ileaf]) {
            lit++;
        }
    }

    if (lit>15) {
        hot_total++;
        hot[iloop]=1;
    }
}

float percent_hot = ( (float) hot_total ) / ((float) nevent);
cout<<"Hot Events / Total Events = "<<hot_total<<" / "<<nevent<<" = "<<percent_hot<<endl;
cout<<"Events considered (not 'hot') = "<<(nevent - hot_total)<<endl<<endl;

//*****

int stepsx = 12;
//int stepsy = 12;
float za[12] = {.5 ,.75, 1.0, 1.25, 1.5, 1.75, 2.0, 2.25, 2.5, 2.75, 3.0, 3.25};
//float a[12] = {};
//float zb[12] = {-.75, -.5, -.25, 0.0, .25, .5, .75, 1.0, 1.25, 1.5, 1.75, 2.0};
//float zb[2] = {-0.842, -0.524, -0.253, 0.0, 0.253, 0.524, 0.842, 1.282};
float a[12][31];
// float b[12][31];

//Set thresholds_a
for (int ihist=1; ihist<=30; ihist++) {
    for (int i=0; i<stepsx; i++){
        a[i][ihist] = za[i]*SPE_mean[ihist];
        //a[i][ihist] = SPE_sigma[ihist]*za[i]+SPE_mean[ihist];
        //b[i][ihist] = SPE_sigma[ihist]*zb[i]+SPE_mean[ihist];
    }
}

//Efficiency
int trig1[8][12]; //N
int trig2[8][12]; //k
float err1[8][12]; //Poisson
float err2[8][12]; //Binomial
float eff[8][12];

for(int i=0; i<8; i++) {
    for(int j=0; j<12; j++) {
        trig1[i][j] = 0;
        trig2[i][j] = 0;
        err1[i][j] = 0.0;
        err2[i][j] = 0.0;
        eff[i][j] = 0.0;
    }
}

```

```

    }
}

for (int iloop=0;iloop<nevent;iloop++){
    tv__tree->GetEntry(iloop);

    for (int imid = 12; imid<=19; imid++) {

        //Exclude the hot events
        if (hot[iloop] != 1){

            //Loop over all changes in thresholds
            for (int ia=0; ia<12; ia++) {

                //Are scintillators in column (top and bottom row) above the new and improved
                threshold_a?
                if ( (l_s[imid-10]->GetValue() > a[ia][imid]) && (l_s[imid+10]->GetValue() >
a[ia][imid]) ){

                    trig1[imid-12][ia]++;

                    //Are scintillators in column (middle row) above the old threshold?
                    if ( (l_s[imid-1]->GetValue() > thresholds[imid-1]) || (l_s[imid]->GetValue() >
thresholds[imid]) || (l_s[imid+1]->GetValue() > thresholds[imid+1]) ){

                        trig2[imid-12][ia]++;
                    }
                }
            }
        }
    }

    for(int i=0; i<8; i++) {
        for(int j=0;j<12;j++) {
            float k = float(trig2[i][j]);
            float N = float(trig1[i][j]);
            eff[i][j] = k/N;
            err1[i][j] = k/N * sqrt(1/k + 1/N);
            err2[i][j] = 1/N * sqrt(k*(1 - k/N));
        }
    }

    for (int j=0; j<12; j++) {
        for (int i=0; i<8; i++) {
            cout<<setw(7)<<setiosflags(ios::fixed)<<setprecision(4)<<eff[i][j];
        }
        cout<<endl;
        for (int i=0; i<8; i++) {
            cout<<setw(7)<<trig1[i][j];
        }
        cout<<endl;
        for (int i=0; i<8; i++) {
            cout<<setw(7)<<setiosflags(ios::fixed)<<setprecision(4)<<err1[i][j];
        }
        cout<<endl;
        for (int i=0; i<8; i++) {
            cout<<setw(7)<<setiosflags(ios::fixed)<<setprecision(4)<<err2[i][j];
        }
        cout<<endl<<endl;
    }

}
return 0;
}

```

Efficiency_ab.C

```
#include <iostream>
#include <iomanip>

int Efficiency_ab()
{
    //Backup to file
    stringstream *output_buf;
    ofstream output_stream;
    output_stream.open("Efficiency_ab.txt");
    output_buf = output_stream.rdbuf();
    cout.rdbuf(output_buf);

    //Input:
    // get 'thresholds[]' and 'SPE_mean[]/sigma[]' from 'SPE_P3.C'

    //Program:
    // loops over P1 through P7 calculating single channel efficiency of scintillators 12-19
    // the effect of overlap is taken into consideration with an 'or' statement

    gROOT->ForceStyle();
    gROOT->SetStyle("Plain");

    //Determined by inspection of the minimum for P3
    float thresholds[32] = {0.0, 0.0, 70.0, 70.0, 70.0, 70.0, 90.0, 90.0, 100.0, 100.0, 100.0,
                          80.0, 90.0, 80.0, 80.0, 90.0, 80.0, 70.0, 80.0, 80.0, 50.0,
                          40.0, 40.0, 60.0, 60.0, 30.0, 80.0, 40.0, 50.0, 90.0, 80.0, 0.0};

    //SPE data from SPE_P3.C
    float SPE_mean[32] = {0.0, 535.54, 164.95, 164.60, 156.23, 158.08, 242.23, 173.04, 220.08, 227.65, 189.09,
                        150.40, 178.75, 146.53, 166.79, 165.97, 194.37, 145.65, 208.20, 176.26, 129.30,
                        125.46, 116.04, 137.64, 164.51, 83.40, 159.28, 113.11, 132.74, 185.54, 178.19, 0.0};

    float SPE_sigma[32] = {0.0, 147.01, 64.25, 53.30, 56.86, 66.92, 100.89, 60.87, 76.10, 84.19, 85.30,
                          58.49, 66.30, 49.03, 59.69, 57.57, 77.03, 57.17, 102.89, 94.51, 60.08,
                          53.61, 51.55, 51.64, 61.61, 45.14, 61.86, 61.10, 56.97, 86.76, 83.53, 0.0};

    gStyle->SetPalette(1);

    //Allocate array indicating whether or not an event is 'hot' outside the loop as a pointer
    int* hot = NULL;

    //Loop over P1.root through P7.root
    for (int ifile=3; ifile<=3; ifile++){

        string infile = Form("P%d.root",ifile);
        TFile *_file0 = TFile::Open(infile.c_str());

        cout<<endl<<endl<<"For "<<infile<<": "<<endl;

        //Get the tree
        tv__tree = (TTree *) gROOT->FindObject("ntuple");

        //Get the leaves
        TLeaf *l_s[32];
        l_s[0] = tv__tree->FindLeaf("bottomtrig");
        l_s[31] = tv__tree->FindLeaf("toptrig");
        for ( int ileaf = 1; ileaf <= 30; ileaf++ ) {
            string s_leaf =Form("s%d",ileaf);
            l_s[ileaf] = tv__tree->FindLeaf(s_leaf.c_str());
        }

        // 'nevent' is the number of events
        Int_t nevent = int(tv__tree->GetEntries());

        //*****

```

```

// 'hot[event#]==1' iff the event is hot (i.e. greater than than 50percent of scint. above
threshold)
// requires some squirrely programming due to inability to redim an array in C++
delete [] hot;
hot = NULL;
hot = new int[nevent];
for (int ihot=0; ihot<nevent; ihot++) {
    hot[ihot] = 0;
}
int lit=0;
int hot_total=0;

for ( int iloop = 0; iloop < nevent; iloop++) {
    tv__tree->GetEntry(iloop);

    lit = 0;
    for (int ileaf=1; ileaf<=30; ileaf++) {
        if ( l_s[ileaf]->GetValue() > thresholds[ileaf]) {
            lit++;
        }
    }

    if (lit>15) {
        hot_total++;
        hot[iloop]=1;
    }
}

float percent_hot = ( (float) hot_total ) / ((float) nevent);
cout<<"Hot Events / Total Events = "<<hot_total<<" / "<<nevent<<" = "<<percent_hot<<endl;
cout<<"Events considered (not 'hot') = "<<(nevent - hot_total)<<endl<<endl;

//*****

int stepsx = 8;
int stepsy = 8;
//float za[12] = {-0.5 ,0.0, 0.5, 1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0, 4.5, 5.0};
//float zb[12] = {-0.75, -0.5, -0.25, 0.0, .25, .5, .75, 1.0, 1.25, 1.5, 1.75, 2.0};
float za[8] = {-0.842, -0.524, -0.253, 0.0, 0.253, 0.524, 0.842, 1.282};
float zb[8] = {-0.842, -0.524, -0.253, 0.0, 0.253, 0.524, 0.842, 1.282};
float a[12][31];
float b[12][31];

//Set thresholds_a
for (int ihist=1; ihist<=30; ihist++) {
    for (int i=0; i<stepsx; i++){
        a[i][ihist] = SPE_sigma[ihist]*za[i]+SPE_mean[ihist];
        //b[i][ihist] = SPE_sigma[ihist]*zb[i]+SPE_mean[ihist];
    }
}

//Set thresholds_b
for (int ihist=1; ihist<=30; ihist++) {
    for (int i=0; i<stepsy; i++){
        //a[i][ihist] = SPE_sigma[ihist]*za[i]+SPE_mean[ihist];
        b[i][ihist] = SPE_sigma[ihist]*zb[i]+SPE_mean[ihist];
    }
}

//Efficiency
int trig1[8][8];
int trig2[8][8];

for (int imid = 12; imid<=19; imid++) {

    //Ensure trig counters initialized to zero
    for (int iclearx=0; iclearx<stepsx; iclearx++){

```

```

    for (int icleary=0; icleary<stepsy; icleary++){
        trig1[iclearx][icleary] = 0;
        trig2[iclearx][icleary] = 0;
    }
}

for (int iloop=0;iloop<nevent;iloop++){
    tv__tree->GetEntry(iloop);

    //Exclude the hot events
    if (hot[iloop] != 1){

        //Loop over all changes in thresholds
        for (int ia=0; ia<stepsx; ia++) {
            for (int ib=0; ib<stepsy; ib++){

                //Are scintillators in column (top and bottom row) above the new and improved
                threshold_a?
                if ( (l_s[imid-10]->GetValue() > a[ia][imid-10]) && (l_s[imid+10]->GetValue() >
                a[ia][imid+10]) ){

                    trig1[ia][ib] = trig1[ia][ib] + 1;

                    //Are scintillators in column (middle row) above the old threshold?
                    if ( (l_s[imid-1]->GetValue() > b[ib][imid-1]) || (l_s[imid]->GetValue() >
                    b[ib][imid]) || (l_s[imid+1]->GetValue() > b[ib][imid+1]) ){

                        trig2[ia][ib] = trig2[ia][ib]+1;
                    }
                }
            }
        }
    }
}

cout<<"FOR s"<<imid<<":"<<endl;

//Print Denominator (Number of events satisfying first trigger)
for (int ix=0; ix<stepsx; ix++){
    for (int iy=0; iy<stepsy; iy++) {
        cout<<setw(7)<<setiosflags(ios::right)<<trig1[ix][iy];
    }
    cout<<endl;
}
cout<<endl;

//Print Efficiency
for (int ix=0; ix<stepsx; ix++){
    for (int iy=0; iy<stepsy; iy++) {

        //Print efficiency provided trig1 is not zero
        if (trig1[ix][iy] != 0) {
            float eff = ((float) trig2[ix][iy]) / ((float) trig1[ix][iy]);

            cout<<setw(7)<<setiosflags(ios::fixed)<<setiosflags(ios::right)<<setprecision(4)<<eff;
            cout<<",";
        }
        if (trig1[ix][iy] == 0) {
            cout<<setw(7)<<"ERR";
        }
    }
    cout<<endl;
}
cout<<endl<<endl;
}
}
return 0;
}

```

Surface.C

```
#include <iostream>
#include <iomanip>

int Surface()
{
    //Input:
    //

    //Program:
    //
    //Output:
    // plots of 2D histograms with linear fit

    //Backup to file
    stringstream *output_buf;
    ofstream output_stream;
    output_stream.open("Surface.txt");
    output_buf = output_stream.rdbuf();
    cout.rdbuf(output_buf);

    gROOT->ForceStyle();
    gROOT->SetStyle("Plain");
    gStyle->SetOptFit(1100);
    gStyle->SetPalette(1);

    float matrix12[12][12]={0.9631, 0.9523, 0.9433, 0.9283, 0.9127, 0.8981, 0.8786, 0.8553, 0.8077,
0.7325, 0.6546, 0.5750,
    0.9635, 0.9527, 0.9434, 0.9284, 0.9133, 0.8990, 0.8797, 0.8560, 0.8084, 0.7324, 0.6533, 0.5749,
    0.9642, 0.9532, 0.9440, 0.9289, 0.9141, 0.8994, 0.8795, 0.8551, 0.8076, 0.7335, 0.6535, 0.5739,
    0.9642, 0.9540, 0.9448, 0.9296, 0.9156, 0.9015, 0.8813, 0.8570, 0.8113, 0.7372, 0.6569, 0.5759,
    0.9633, 0.9530, 0.9440, 0.9294, 0.9163, 0.9021, 0.8816, 0.8579, 0.8114, 0.7364, 0.6567, 0.5758,
    0.9639, 0.9545, 0.9451, 0.9304, 0.9172, 0.9029, 0.8824, 0.8578, 0.8111, 0.7354, 0.6575, 0.5789,
    0.9645, 0.9556, 0.9463, 0.9319, 0.9187, 0.9044, 0.8836, 0.8595, 0.8117, 0.7347, 0.6568, 0.5772,
    0.9663, 0.9582, 0.9483, 0.9344, 0.9204, 0.9056, 0.8845, 0.8589, 0.8103, 0.7357, 0.6602, 0.5811,
    0.9699, 0.9613, 0.9513, 0.9377, 0.9262, 0.9106, 0.8905, 0.8649, 0.8167, 0.7413, 0.6655, 0.5866,
    0.9687, 0.9588, 0.9496, 0.9348, 0.9244, 0.9103, 0.8918, 0.8679, 0.8187, 0.7462, 0.6742, 0.5993,
    0.9708, 0.9601, 0.9508, 0.9363, 0.9270, 0.9117, 0.8963, 0.8725, 0.8257, 0.7565, 0.6866, 0.6068,
    0.9724, 0.9625, 0.9546, 0.9389, 0.9300, 0.9181, 0.9053, 0.8836, 0.8422, 0.7761, 0.7101, 0.6430};

    // MUST INCLUDE "float matrix13[13][13] = {};" and values from Efficiency_ab.C

    //Book histogram
    TH2F *hist_array = new TH2F("Eff_hist","Leakage and Single Layer Efficiency",8,-(1.0-0.15),-
(1.0-0.95),5,0.15,0.65);

    for (int ihist=0; ihist<8; ihist++) {

        //THIS IS REALLY UGLY
        float matrix_temp[8][8];
        if (ihist == 0) {
            for (int ix=0; ix<8; ix++){
                for (int iy=0; iy<8; iy++){
                    matrix_temp[ix][iy] = matrix12[ix][iy];
                }
            }
        }

        // MUST INCLUDE A 'matrix_temp[8][8] = matrix1X[12][12]' for each middle scintillator'

        //And Finally...
        float matrix[8][5];
        for (int ix=0; ix<8; ix++){
            for (int iy=0; iy<5; iy++){

                matrix[ix][iy] = matrix_temp[ix][iy];
            }
        }
    }
}
```

```

        float xvar = -(1.0 - (float(0.1*ix) + 0.2));
        float yvar = float(0.1*iy) + 0.2;
        float zvar = matrix[ix][iy] / 8.0;

        hist_array->Fill(xvar,yvar,zvar);
    }
}

/*
//Book Graphs
TGraph2D *graph_array[8];
for (int igrph=0; igrph<8; igrph++) {
    graph_array[igrph] = new TGraph2D(64);
}
*/

TF2 *func_array = new TF2("ffit","[0]+[1]*x+[2]*y",-1.0,0.0,0.0,1.0);
func_array->SetParameters(.988,0.0135,-.104);
hist_array->Fit(ffit,"0");

// MUST INCLUDE AN ARRAY OF FUNCTIONS IF MULTIPLE FITS DESIRED

for (int ifunc=0; ifunc<=0; ifunc++) {
    float parameter[3];
    float error[3];
    for (int ipar=0; ipar<3; ipar++) {
        parameter[ipar] = func_array->GetParameter(ipar);
        error[ipar] = func_array->GetParError(ipar);
    }
    cout<<"Parameters drawn from averaged efficiencies"<<endl;
    cout<<parameter[0]<<" w/ error "<<error[0]<<" is the constant"<<endl;
    cout<<parameter[1]<<" w/ error "<<error[1]<<" is the change in efficiency per fraction SPE of
the trigger scintillators"<<endl;
    cout<<parameter[2]<<" w/ error "<<error[2]<<" is the change in efficiency per fraction SPE of
the middle scintillators"<<endl<<endl;
}

////Print histograms
TCanvas *c_1 = new TCanvas("c_1","Efficiencies",0,0,500,500);

// c_1->Divide(4,2,0.04,0.04);
for (int ipad=1; ipad<=1; ipad++) {
    c_1->cd(ipad);
    func_array->Draw("SURF4");
    hist_array->Draw("SURF2 same");
    func_array->GetXaxis()->SetTitle("Thresh 'a' (Frac_SPE)");
    func_array->GetXaxis()->SetLabelSize(.03);
    func_array->GetXaxis()->SetTitleOffset(1.6);
    func_array->GetXaxis()->SetNdivisions(505);
    func_array->GetYaxis()->SetNdivisions(505);
    func_array->GetYaxis()->SetTitle("Thresh 'b' (Frac_SPE)");
    func_array->GetYaxis()->SetLabelSize(.03);
    func_array->GetYaxis()->SetTitleOffset(1.6);
    //hist_array[ipad-1]->GetZaxis()->SetTitle("Efficiency");
    func_array->GetZaxis()->SetLabelSize(.03);
    func_array->GetZaxis()->SetTitleOffset(1.6);

    //func_array->Draw("SURF1 same");
}

//c_1.Print("graphs.gif");

return 0;
}

```