An Algorithm to Resolve Dead Blocks in the Radiative-ϕ Experiment Software

A senior project submitted in partial fulfillment of the requirements

for a

Bachelor of Science in Physics

from the College of William & Mary in Virginia,

by

Adam P. Gurson

Project Advisor: Dr. David Armstrong

Contents

1 INTRODUCTION

There are two scalar mesons [1], the isoscalar $f_0(975)$ and the isovector $a_0(980)$, which lately have been receiving a great deal of attention from many particle physicists. Having been incorporated into the particles of the subatomic realm, they were long thought to belong to the ${}^3P_0$ $\bar{q}q$ scalar nonet. However, there have been many subtle inconsistencies with this scenario, and the designation has come under heavy inquiry. For instance, the $f_0(975)$ and the $a_0(980)$, are fairly light particles, having respective masses of 975 MeV/c^2 and 980 MeV/c^2. It is unusual for them to be placed in this category when its strange member is the lightest scalar kaon, the $K_0^*(1430)$, having a mass of 1430 MeV/c^2. While looking for a more suitable model, these light-weight particles have been suggested to be 4-quark states ($q^2\bar{q}^2$) or even effective molecules ($K\bar{K}$), and Quantum Chromo-Dynamics does allow the existence of such particles. While many different models for these particles have been proposed, no one model has proven conclusive.

The purpose of this computer software-based research is to aid in the evaluation and interpretation of data to be collected in the next phase of an experiment currently in production in Hall B of Jefferson Laboratory by the Radiative-ϕ group. The primary goal of the experiment is to empirically determine the branching ratios of the two decays

$$\phi \to \gamma f_0 \to \gamma\pi\pi \tag{1}$$

and

$$\phi \to \gamma a_0 \to \gamma\eta\pi. \tag{2}$$

It is theorized [2] that different models for the f_0 and a_0 will yield a different result for the ratio of these respective branching ratios. If an empirical ratio is determined, true insight into the quark-gluon makeup of the f_0 and a_0 may be achieved.

The software implemented in this research will allow real-time and post-evaluation of the data collected. By using a graphical interface designed to emulate the real components of the detector used in the experiment, the software provides greater ease of interpretation and manipulation of the Radiative-ϕ experiment.

2 BASIC THEORY

The makeup of the f_0 and a_0 could be solely quark-based (standard model or 4-q bag), solely gluon-based (a "glueball"), a hybrid of the two (such as a $q\bar{q}g$), or even a mesonic molecule ($K\bar{K}$). Each of these models would necessarily behave differently in reference to branching ratios under the Okubo-Zweig-Iizuka (OZI) rule [3]. Several calculations have been performed [2,4] to predict the ratio

$$\frac{BR(\phi \to a_0\gamma)}{BR(\phi \to f_0\gamma).} \tag{3}$$

For instance, if the f_0 were a quark-based particle such as $\bar{s}s$, its radiative decay branching ratio is expected to be $\sim 10^{-5}$. Now, as Ref. [2] suggests, it is possible to compare this model against one where the f_0 is a glueball, and, to follow OZI-violation rules, this requires the glueball-quark mixing angle to be less than $[\Gamma(f_0 \to \pi\pi) \,/\, \Gamma(^3P_0 \to \pi\pi)]^{\frac{1}{2}}$. Simplifying:

$$\begin{aligned} \Gamma(\phi \to (\text{glueball})\gamma) &\leq \frac{\Gamma(f_0 \to \pi\pi)}{\Gamma(^3P_0 \to \pi\pi)} \cdot \Gamma(\phi \to f_0(\text{quark})\gamma) \\ &\leq \frac{1}{20} \cdot \Gamma(\phi \to f_0(\text{quark})\gamma), \end{aligned}$$

shows that we would expect the branching ratio to be an order of magnitude less for the glueball model than for the standard quark model. Continuing this method, predicted values, such as those found in Table 1 [5], have been obtained.

model	$\frac{BR(\phi\to a_0\gamma)}{BR(\phi\to f_0\gamma)}$
quark model	~ 0
$K\bar{K}$ molecule	1
4-q bag	9

Table 1: Model predictions for ratios of radiative ϕ-decay couplings.

While a large amount of theoretical work has been accomplished, conclusive empirical data on the true branching ratios is still lacking. Some preliminary measurements [4, 6] have been made by other groups along with accompanying analysis of the respective results, however the next phase of the Radphi experiment is anticipated to be more enlightening.

3 THE RADPHI EXPERIMENT

As mentioned above, the primary goal of the Radphi experiment is to empirically determine the branching ratios sought after in section 2 so that a better understanding of the f_0 and a_0 makeup can be achieved [7]. The experiment can best be described as consisting of three main parts: the creation of the ϕ particles, the main detector, and the accompanying software package, which is the primary focus of my research. To create an influx of ϕ particles, the experiment utilizes the Bremsstrahlung process to create a "tagged" photon beam from Jefferson Lab's continuous electron beam.

This photon beam is then incident upon a beryllium target, allowing the photons to interact with the protons inside the beryllium. The reaction $\gamma p \to \phi p$ can then provide the appropriate ϕ particles, which then go on to decay as expected. The ϕ particle has several different decay sequences, most of which are more common than those from section 2. It is therefore necessary to discriminate between decay sequences. Again, the two decays of interest are

$$\phi \to \gamma f_0 \to \gamma \pi^0 \pi^0 \to \gamma\gamma\gamma\gamma\gamma \tag{4}$$

and

$$\phi \to \gamma a_0 \to \gamma \eta \pi^0 \to \gamma\gamma\gamma\gamma\gamma. \tag{5}$$

Therefore, a relevant decay consists ultimately of 5 photons and a proton. To detect these properly, the detector consists of three main parts: barrel scintillator detector (BSD), the charged particle veto wall (CPV), and the lead glass detector (LGD).

An experimental "event" consists of a time-slice of particle interactions between the separate detectors. Ideally, all interactions will be the result of one ϕ decay. The BSD is a barrel of 48 scintillator counters in 3 layers surrounding the beryllium target. For an event to be valid, a recoil proton from the $\gamma p \to \phi p$ decay must be detected by one of the scintillators in each layer.

The LGD is a series of 784 lead glass blocks, 628 of which are connected to a photomultiplier tube sitting downstream of the BSD. The purpose of the LGD is to detect the 5 remaining photons from the ϕ decay (actually, by this time, the 5 remaining photons have themselves radiated into a "shower" of many localized photons [8] which are to then be reconstructed by means of the algorithm to be

described in section 5). The LGD is used to detect the energy and spatial vector coordinates of the incident photons.

In an attempt to protect the experiment from incident charged particles which can interfere with the LGD, the CPV is placed just upstream of the LGD. It is designed to detect any charged particles which have entered the beam area and may falsely interact with the LGD, yielding misleading results.

Clearly, the lifetime of the f_0 and a_0 are too short ($\sim 10^{-23}$ s) to detect these particles directly. All of the data of interest consists of only the 5 photons which are left behind. It is therefore necessary to "rebuild" these photons into a "pre-decay" particle and establish their connection with either an f_0, a_0, some particle corresponding to their respective decays, or none of these. In essence, using conservation of energy and momentum, the photons detected by the LGD are to be combined until their total energy resembles the sought after particle. This "rebuilding" method has already been successfully used in other similar experiments [4, 6, 9] and some preliminary results for the Radphi experiment based on this technique are available [5].

Once all of these ϕ-decays have been properly identified, it will be possible to empirically derive real values for $BR(\phi \to f_0\gamma)$ and $BR(\phi \to a_0\gamma)$, thereby discriminating among the many proposed models for the f_0 and a_0.

4 THE SOFTWARE

The data collected during the experiment are initially in a very terse, encoded state, which is not easy to directly comprehend, consisting of calibration constants, various

detector channel hits, and other information relevant to specific run setups and events. The purpose of the software is to aid the researcher in reorganizing this data into individual records from which it can be easier to grab the necessary information, as well as providing the user with a graphical interface designed to recreate any specific recorded event. The graphical interface can be used to evaluate the data quickly while it is being obtained. It uses special algorithms to quickly accomplish much of the photon "rebuilding" process, allowing the researcher to easily identify initial decay sequences as well as faulty data, indicating a possible need for detector readjustment.

The software is a suite of programs written in ANSI C which was created by many different people at different times for different needs. Since its conception, the code has grown, and the different parts of the code have been incorporated together into what is now to be used for the next phase of the Radphi experiment. This new version consists of an analytical side and an interactive side. The analytical side is designed to handle most of the "grunt" work such as transforming the initial mess of input data into simple descriptive groups of human-readable information. The interactive side is designed to graphically emulate each of the main detectors, allowing the researcher to evaluate and manipulate the data in real-time.

Data from the Radphi experiment is stored in three main forms. The initial form is the compact and terse raw data already described. This is taken and run through a preprocessor program which "wraps" parts of the raw data with a special C data structure. This data is considered packed and cannot be directly read by the graphical interface. The packed data can then be run through an unpacking program which collects the data into "groups" in the form of C structures relevant to each

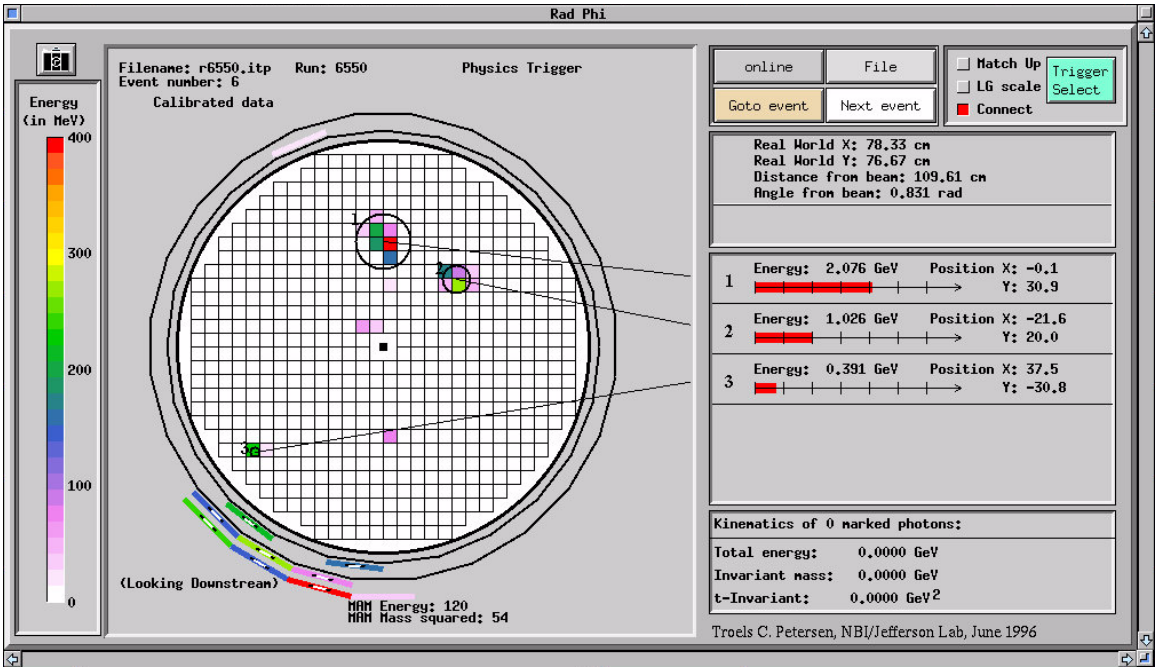

Figure 1: The main window for the graphical interface program.

run and each event within each run. In this unpacked form, the grouped data can be interpreted with greater ease by researchers, but its primary purpose is to now serve as input to the user-interface graphical data-analyzing program. Once given input data for a specific run, this program will display all hits collected on the various detectors by means of a simple color-coding scheme based on energy. Also, once the raw data has been drawn out, the code continues to run a clusterizing algorithm (described in section 5) which attempts to reconstruct the shower left by the 5 final photons. These clusters can then be analyzed and combined directly by the researcher, allowing immediate insight into the data.

Figure 1 shows the main window for the graphical interface program [10]. Each display corresponds to a specific event of a specific run, labeled at the top. For instance, this example is event number 6 of run number 6550. The left side of the

display represents the LGD and the modules inside it which are connected to photomultiplier tubes. The view is looking down the beam (the z-axis). A hit in a photomultiplier tube is represented by a coloring of the corresponding module. The different colors represent the energy of the hit on a scale located to the left. Here we see a few localized photon "showers" have triggered hits on the LGD (a pedestal energy level corresponding to the ambient noise in Hall B is subtracted from all modules to prevent the LGD display from having an overall low-energy pink tint). The clusterizing algorithm will then attempt to reconstruct showers that could have been caused by incident photons. When a cluster is found, a corresponding circle is draw on the display, centered at the hit location of the incident photon (weighted by energy distribution), and with a radius proportional to the energy of the reconstructed photon. The display also allows the user to select and group clusters, whose combined kinematic data is then displayed on the right side of the screen. This method allows for an easy detection of possible incident π^0 or η particles, implying the relevant decay sequences.

The BSD is "rolled out" by the software and shown in Figure 2. There are three lines of detectors, vertical (corresponding to the downstream direction) and both diagonal directions. A colored-in section corresponds to a hit somewhere in the corresponding line of the detector. To pinpoint the location of a recoiled proton, one locates the triangular section of the display where all three directions are colored. Multiple charged particle hits in the detector can further be analyzed for relevance. If no proton is detected, the data for that event could be defective.

As was mentioned above, if a charged particle reaches the LGD, it could trigger

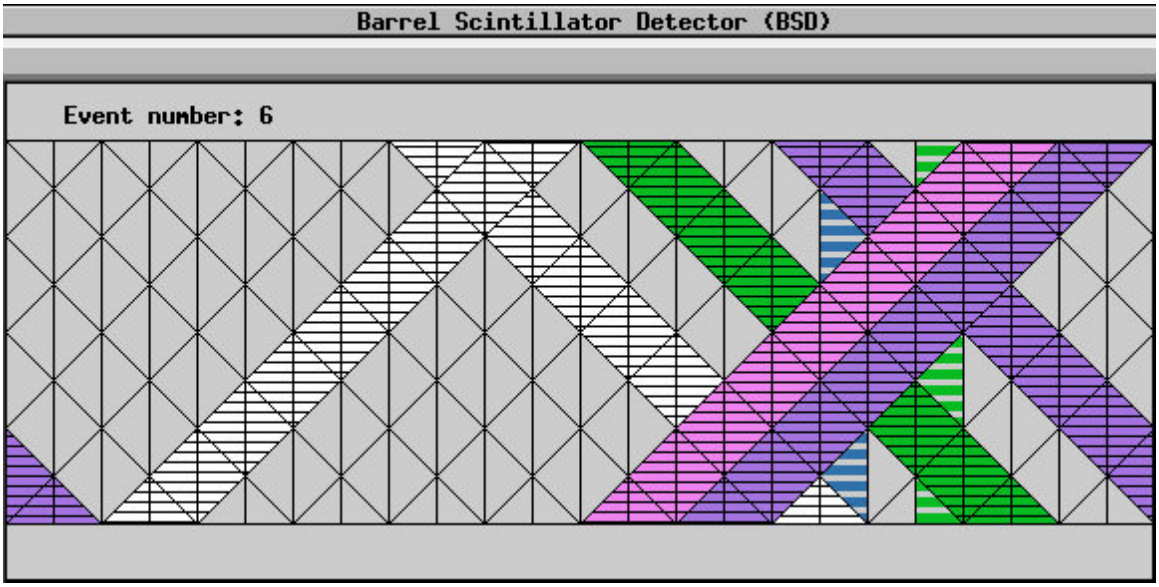

Figure 2: The "rolled out" Barrel Scintillator Detector (BSD).

a misleading hit, hence the presence of the CPV, which is emulated by the computer program as shown in Figure 3. The CPV display is the same size as the LGD display, allowing the researcher to quickly relate information between the two. Color-coded by energy, a hit in one of the CPV target sections (separated by horizontal black lines) corresponds to a possible interfering charged particle. If a strong hit is found in a corresponding section of the LGD, it can most likely be discarded as a charged particle.

Finally, Figure 4 illustrates a display which can be used to track the properties of the incident photon beam. The energy hits drawn correspond to the energies of residual electrons left during the Bremsstrahlung process. This information aids the researcher by predicting the necessary total energy calculated in the reconstruction of the 5 final photons.

Together, this software package resides on the machines at Jefferson Lab and can

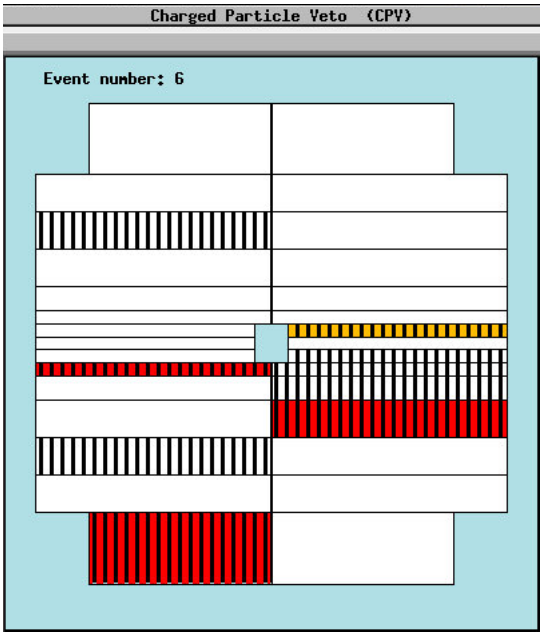

Figure 3: The computer depiction of the CPV.

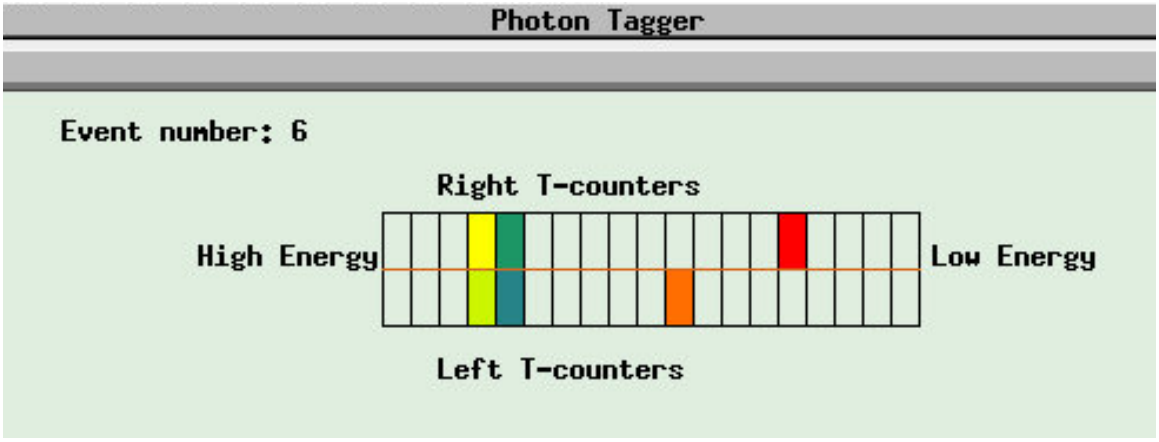

Figure 4: The computer window dealing with the tagged photon beam.

be used for direct analysis of the data during experimentation or for post-analysis purposes.

5 THE CLUSTERIZER

It has already been established that when a photon enters the LGD, it can radiate into a shower of low-energy photons whose total energy is equal to the energy of the incident photon. This occurs because the incident photon creates an electron-positron pair. This electron then, through the Bremsstrahlung process, radiates another photon, which then radiates another pair, and so on. This can again be seen by noting the cluster of colored squares in the upper LGD display in Figure 1. A section of code known as the "clusterizer" attempts to locate these groups of hits and associate them with one or more incident photons. Once these groups are found, their data are combined and marked appropriately on the LGD display.

The clusterizer works in three steps. First, it searches the LGD module by module for the greatest energy hit. It then adds in the 8 surrounding modules and a primary cluster is formed. These modules are then excluded, and the search begins again. This continues until there are no hits greater than 350 MeV (this number was based on an empirical study and appears to generate a reasonable ratio of real to fake clusters).

The next step considers the 16 blocks around any primary clusters. If a hit is found in this area, the module is added to the current cluster. If one of these 16 blocks is shared by two or more clusters, the energy in the module is divided among the sharing clusters in proportion to their energy. Finally, the first step is repeated,

but this time the lower threshold is reduced to 150 MeV.

Once the clusters are formed, the following formula is used to determine their respective "center", or point where the incident photon would have hit had it not dispersed into a shower:

$$x_c = \frac{\sum_{j=1}^{N} w_j(E_j) x_j}{\sum_{j=1}^{N} w_j(E_j)}, \tag{6}$$

where x_c is the assumed x-coordinate of the center, N is the number of modules associated with the cluster, x_j is the x-coordinate of the center of the j-th module, and $w_j()$ is a function of the energy associated with the j-th module designed to create a logarithmic weighting scheme. There is also the appropriate similar equation for the y-coordinate of the center. All of the relevant calculated information is then packed together and added as a new data group associated with the current event. It is this data group which is then read by the LGD display. For more information on the clusterizer, see [8].

6 OUR GOAL

As was mentioned above, the code written for this software project is really a merging, modification, and addendum to different sources of pre-written code. Many changes have been added to aid in the code's usefulness, and more changes are planned for future modification.

Originally, the Event Display program could only accept unpacked, grouped data as input. If packed data was to be analyzed, it had to be preprocessed into unpacked data by a program which would also generate a great deal of correct, yet irrelevant,

data. The actual unpacking code has been incorporated into the Event Display so that if packed data is read, the unpacking will occur automatically without the need for a preprocessor. Also, while the clusterizer had been written, it was implemented for simulated data which used a different format (this was originally written well before real data were available). The clusterizer has now been fully incorporated into the new version of the Event Display, providing relevant data for the LGD display. Finally, there are many different event types which do not correspond to real event data, but instead to calibration information, etc. The code was changed to automatically detect these groups and skip over them when drawing to the Event display (rather than drawing a blank LGD).

While these aforementioned changes have been beneficial, the main focus of this project has been to develop an algorithm which will account for dead LGD blocks. At any given time, a number of the LGD photomultiplier tubes may not be functioning properly. These blocks will not respond to incident photon showers, leaving a blank space in the LGD display. This empty reading is what suggests the name dead blocks.

As mentioned above, when a photon is incident on the LGD, it showers into many readings on separate photomultiplier tubes in different LGD channels. These channels are then clusterized together by the clusterizer described in the previous section to represent groups of energy readings associated with a specific incident photon. If, however, the photon showers into an area which includes dead blocks, some energy associated with that photon will not be read and therefore left out of the cluster. When these energies are then recombined to build up the incident photon, then energy will be too small, creating a skew in the data. Our main goal for this research

is to design and incorporate an algorithm which will go through each of the clusters returned by the clusterizer and look for dead blocks. Once found, the algorithm will use energy readings from surrounding cluster blocks to extrapolate an energy level for any dead blocks. Those dead blocks, along with their new significant energy readings, would then be incorporated into the cluster.

The following sections describe the process involved in searching for dead blocks and the algorithm created to estimate a first-order energy approximation for them.

7 LOCATING DEAD BLOCKS

The first necessary step to extrapolating data into dead blocks is to locate the dead blocks. For this process, a separate algorithm was written. Below is a general description of our algorithm.

While most of the time the LGD is used to detect incident photons, there are other events which occur at regular intervals during an experimental run. These events are not based on photon information, but instead are a type of self-diagnostic which can be used to probe the state of the photomultiplier tubes, electronics, and cables used in the LGD. These events yield data that relate directly to the function of the LGD. While looking for dead blocks, we are interested in two particular types of diagnostic events: Base Test Events, and LGD Monitor Events. The data collected from these two events are crucial to locating dead blocks. Therefore, we describe these events in greater detail below.

7.1 The Base Test Events

Each LGD channel detects incident photon information via a specific photomultiplier tube associated with each channel. This photomultiplier tube works with a chain of dynodes inside, each set to a specific voltage. An incident photon interacts with the photocathode in the tube, which creates a large shower of electrons within the tube. These electrons each create a new shower within the first dynode, and this process continues throughout the length of the tube. The final shower is then detected by the base of the tube, which produces a voltage pulse which can then be digitized by an analog-to-digital converter (ADC). The chain of voltage on the detectors determines the sensitivity of the tube: the higher the voltage placed on the chain, the more sensitive the tube is for detecting incident particles.

The voltages for these tubes are set remotely by a serial line cable which runs into the base of the tubes. It is also possible via this cable to interrogate the base of the tubes to ensure that they are in fact set at the designated voltage. During this test, the base will respond with a pulse containing a signal voltage proportional to the voltage on the base. If, for some reason, the tube is not functioning properly and the voltage on the tube is zero, then the LGD block connected to the photomultiplier tube will be unresponsive to incident photons and should therefore be marked dead. We could therefore use the base test data to look for blocks which return a very low incorrect voltage when interrogated and mark them dead. An example of a base test event and how it appears when viewed with the LGD display is shown in Figure 5.

While it is true that a dead block would show an unusually low reading during

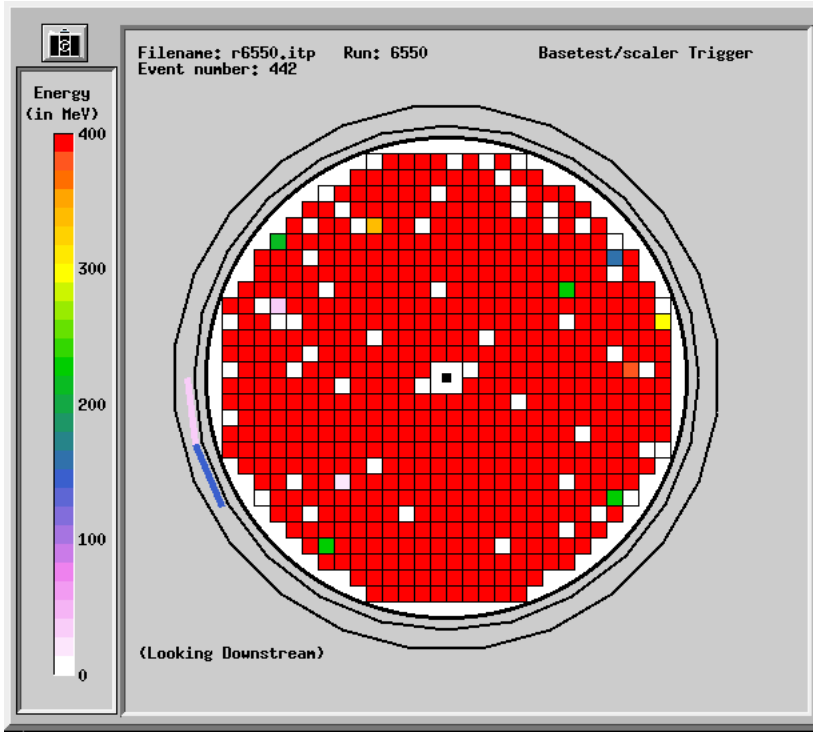

Figure 5: The LGD depiction of a base test event.

this test, we must be careful when making assumptions about the state of *every* block which returns an anomalous reading. This is because that a base could actually fail for other reasons, when, in fact, the photomultiplier tube is functioning perfectly. There are three basic scenarios which could cause a base test to fail, and we now look at each one.

First, we consider the scenario already discussed. We have a photomultiplier tube with a base that is truly dead and therefore cannot detect incident particles. We interrogate the base with the base test signal, and the base correctly returns a very low reading, being dead and having no voltage on it. In this instance, the test has provided us with information useful for correctly identifying dead blocks.

Second, we consider another scenario. The serial cable that collects data from the base of the photomultiplier tube is connected at the other end to an Analog to Digital Converter (ADC), which actually writes records the data into a computer-readable format. It is possible that the base of the tube is actually working perfectly, but the serial cable, or its connection to the ADC, could be malfunctioning. In this case, when probed by a base test, the result would come back as a zero voltage, which is misleading, although still useful for our purposes. This is because even though the photomultiplier tube is collecting data properly, the ADC would not be able to record any data taken by the photomultiplier tube, so the block would still effectively be, and should be marked as, dead.

Finally, it was mentioned above that the base test uses a special independent circuit in the base of the photomultiplier to return a relevant pulse during a base test interrogation. It is possible that this circuit, itself, could be malfunctioning, while the

photomultiplier tube and base, functioning in a completely separate manner, could still be working perfectly. A base test would still fail, however, for the circuitry used to return a signal would not return anything, and the base test would interpret this as a return of zero voltage. This case actually causes a significant problem with using a base test to determine dead blocks. In this scenario, the block is *not* dead, and will function normally during experimentation, but a base test alone would imply the opposite. We have found that this scenario is actually quite common, and a majority of blocks labeled dead by only this method will, in fact, be fine. Therefore, while the base test does provide a good manner of double-checking the status of a block labeled dead, it is still necessary to look for another primary way of identifying dead blocks. We therefore consider the additional method in the next section.

7.2 The LGD Monitor Events

The LGD monitor event is designed to simultaneously subject every photomultiplier tube in the LGD to a fairly uniform shower of light. All blocks that are functioning properly should detect a significant positive energy value, while those blocks not functioning properly will return a value of zero incident energy to the ADC.

Located just in front of the LGD is a sheet of Plexiglas, which is attached to a laser light source. During an LGD monitor event, this laser fires light throughout the Plexiglas which then deflects out and into the LGD, showering light across the entire detector. This light is then detected by the LGD and recorded as data. While the shower should be relatively uniform across the LGD, this is not exactly the case. In the center of the Plexiglas, as with the LGD, is a hole two blocks by two blocks

to allow the particle beam to pass through the equipment. As these corners of the Plexiglas, more light is deflected out and into the LGD. It is therefore likely to expect an increase of overall energy around the center of the LGD which decreases radially. The test is therefore not as uniform as we would like, although it is reasonable to expect blocks which border each other to have similar energy values during this test.

Dead blocks are fairly easy to reliably locate during this test. At any given time, the electronic noise created by the detector itself will create an overall base amount of data in the LGD detectors. This pedestal amount, as it is called, usually has a value of about 50 ADC units. This value is stored into a database, converted to MeV and subtracted from each block automatically for data purposes. Since we are looking at this data to find dead blocks, it is reasonable to assume that any block which yields a value above 50 ADC units during an LGD monitor event is functioning properly, while those blocks which are barely above the pedestal value are most likely dead. Therefore the LGD monitor event is a good way to identify dead blocks. The base test can then be used to verify the information. An example of an LGD monitor event and how it appears when viewed with the LGD display is shown in Figure 6.

7.3 Groups

Before discussing the algorithm, it is important to briefly discuss the concept of groups. A Group is a coding term for a collection of data relevant to a specific concept. Technically, it is really a C structure containing processed data. By wrapping this C structure into a group, the data within the group can be easily retrieved with easy-to-use function calls, and a majority of the C notation can then be hidden from the

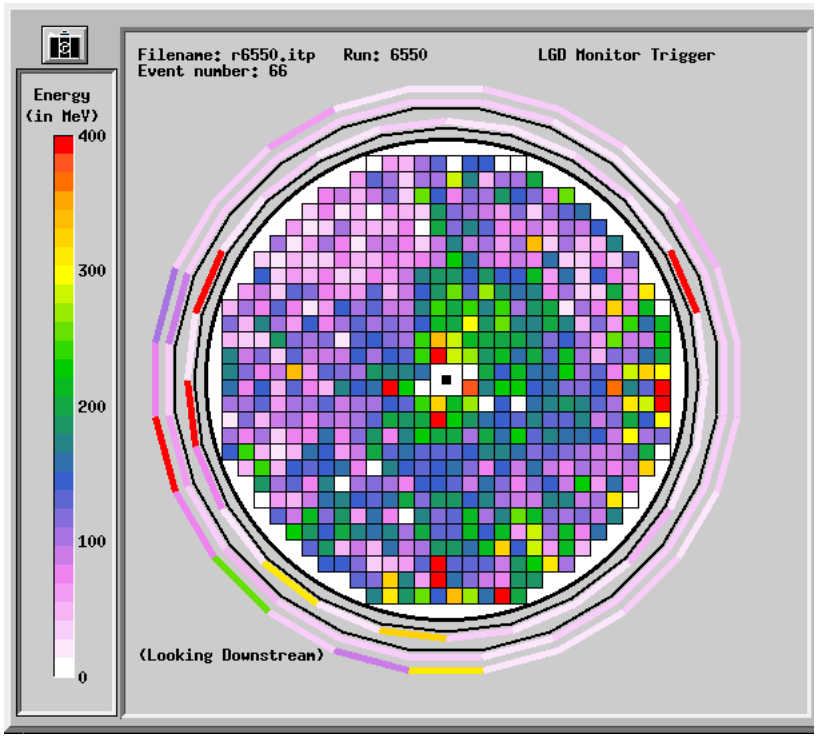

Figure 6: The LGD depiction of an LGD monitor event.

user.

Our dead block location algorithm relies on the information stored in a specific group: `GROUP_LGD_ADCS`. When energy information for a particular event is sent from the LGD to the ADC, the ADC writes out energy information for each channel/block in the LGD. It is this energy information, sorted by channel, which is then put into a C structure and converted into the Group `GROUP_LGD_ADCS`. Therefore, this group contains a C structure that contains, for every channel in the LGD, a field designating the LGD channel number and another field containing the energy for that channel. While the finer details of this group and groups in general are not crucial to the understanding of the algorithms described here, it is important to realize that the dead block algorithm is looking for energy and channel information to be in this particular form.

7.4 The Algorithm

As was mentioned above, a given run of the experiment can contain many types of data. It could be real event data, such as energy hits due to incident photons, or diagnostic data such as base test and LGD monitor event readings. Our dead block algorithm is only concerned with base test and LGD monitor events. Also, the algorithm will only be run once per run number, so that any dead blocks found will be labeled dead throughout the entire course of one run.

The algorithm begins for a given run number by scanning through the list of events until it finds either a base test event or LGD monitor event. If it finds an event of either type, it then locates the `GROUP_LGD_ADCS` group and reads in the energy value

associated with each channel for that event.

If this is a base test event, each block is checked to see whether or not it has returned an energy value of 5 ADC units or less. If this is the case for a given block, we assume the block to be dead during the duration of this event for reasons stated in section 7.1. If this is an LGD monitor event and a given channel has returned an energy value between 40 and 55 ADC units, inclusive, we assume the block to be dead during the duration of this event for reasons stated in section 7.2.

It is now a good place to introduce the concept of base test points and LGD monitor points. We will be looking over several base test and LGD monitor events before any overall dead block decisions are made, so we keep a count of how many times a given channel has been selected as dead for all of the tests. This is a point count for each channel in each type of test. For example, if the algorithm is currently scanning through the channels in a base test and channel 143 fulfills the criterion of dead block, 1 base test point is added to block 143's counter. A separate count is made for LGD monitor events. Our algorithm continues this process until exactly 5 base test events and 5 LGD monitor events have been scanned for dead blocks.

After the algorithm has searched through all of the relevant events, it is actually time to count up all of the points for each channel and determine which blocks should be labeled dead for the entire duration of the run number. Let n be the number of base test points a given channel has accumulated and m the number of LGD monitor points a given channel has accumulated. The algorithm once again goes through each channel in the LGD, and a block is labeled dead only if

$$\alpha n + \beta m \geq c, \tag{7}$$

where α and β are integer weighting constants for each event type and c is a small integer. For our research, we have found that good values are

$$\alpha = 1, \ \beta = 2, \ \text{and} \ c = 9. \tag{8}$$

In other words, we weight finding a dead block during a LGD monitor event to be twice as significant as finding a dead block in a base test event. This is because many of the base test events can fail due to poor test circuitry in the photomultiplier base. This would cause the block to appear dead when, in fact, this is not the case.

Any block fitting the criterion of equation (7) is labeled dead, and this information is stored in a size 784 bitmap array (1 slot for each channel). If the block is dead, a 1 is entered in the block's respective slot, otherwise a 0 is entered. This array is kept in the Event Display code and is used to allow the creation of a new display button: Dead Bks. When pushed, this button causes the location of all dead blocks to be shown on the LGD display by drawing a slash through the relevant blocks.

Currently, this dead block algorithm is embedded completely within the Event Display code. This forces the algorithm to be rerun every time a set of events for a given run number is displayed. Eventually, this code will be moved to another location, where it will be run *before* the LGD Display is used. For this reason, the algorithm also stores dead block information by run number into a mapmanager database. Again, this will eventually be the preferred means of retrieving dead block information. For now, users using the LGD Event Display will notice a 20 to 30 second delay while a run number file loads. This covers the time needed for the algorithm to find and scan 5 base tests and 5 LGD monitor events as well as to write

out the dead block information.

8 THE 1ST ORDER DEAD BLOCK ADJUSTMENT ALGORITHM

Once we have located the dead blocks for a given run, the goal is to use the information in an algorithm that will estimate a probable energy value for any dead blocks located within a cluster for a given event. Clusters can literally be of any shape, so it is difficult to create a general algorithm to solve this problem. It was therefore the goal of this research to begin by providing a first-order correction to the problem. By a first order correction, we mean only attempting to extrapolate values for dead blocks located within the 8 blocks surrounding the center of the cluster. We assume that a correction to these blocks would yield the most significant positive change in the data. The following is a description of our 1st order correction algorithm.

8.1 Required Input and Provided Output

The algorithm which makes first order corrections to the dead blocks in clusters is basically one modular C function. This section describes the input parameters that the algorithm needs for execution as well as the output parameters returned by the algorithm upon its completion.

Upon its completion, the clusterizer returns two types of C structures relevant to the clusterized data: an `lgd_hits_t` structure and an `lgd_clusters_t` structure. The `lgd_hits_t` structure effectively contains two types of information. First, it contains

a variable, `nhits`, which holds the total number of channels present in *any* cluster for the given event. Also, it contains a list of every block number and its respective energy reading, sorted by cluster. For example, if `nhits` is 20, then 20 blocks of the LGD are part of clusters for this event, and there is a list of 20 relevant block numbers and their respective energies, all contained within the `lgd_hits_t` structure. As a final note on this structure, if a block is part of more than one cluster, then it is counted twice in `nhits` and it is listed twice in the blocks list. This information is clearly relevant, because it contains all energy information necessary to extrapolate energies for dead blocks.

Second, we have the `lgd_clusters_t` structure. The `lgd_hits_t` list of information is helpful, but all of the hit information is stored in one structure and therefore runs together. There is no way to look at this list by itself and determine how many clusters there are in the list and where the information for one ends and the next one begins. This is where the `lgd_clusters_t` structure comes in. It contains information relevant to each cluster, and where to look in the `lgd_hits_t` structure for the proper information. The `lgd_clusters_t` structure contains information such as how many total clusters there are, stored in the variable `nClusters`. Then for each cluster, it contains the following information: the number of LGD blocks used in the cluster, an index into the `lgd_hits_t` list that states where the first block for this cluster can be found, flags (which can be set to certain values to tag certain clusters), a vector stating the x, y, and z coordinates of the center of the cluster, the width of the cluster (which is a variable parameterizing a given distance from the center in which most of the cluster blocks can be found), and the total energy of the cluster (which is simply

the sum of all relevant energies in the `lgd_hits_t` list).

The information from these structures is usually wrapped into a group format immediately upon its return from the clusterizer. However, our algorithm must deal with this information directly, so this data should not be made into a group until after our algorithm has run. Basically, our code has two input parameters, and it returns two output parameters. The two input parameters should be a pointer to the `lgd_hits_t` structure and a pointer to the `lgd_clusters_t` structure which have been returned by the clusterizer. Our algorithm will then use this information along with the information read in from the dead block database to find 1st order dead blocks near clusters, give them an extrapolated energy, and then add them to the clusters as described in the next section. Our algorithm will therefore return two *new* structures, one a `lgd_hits_t` and the other a `lgd_clusters_t`. The user should pass in a *pointer* to `NULL` pointers of the proper type, and the algorithm will then attach the relevant information to them.

The new `lgd_hits_t` structure will contain block number and energy information for all of the old cluster blocks as well as information for any new blocks that were once dead but now added to the cluster. This information will be properly sorted by cluster. The new `lgd_clusters_t` structure will be similar to the old `lgd_clusters_t` structure, except for 3 fields. If the algorithm adds blocks to a cluster, the field containing the number of blocks for a given cluster will be incremented accordingly. Also, the addition of new blocks for a cluster would most likely shift the list in the `lgd_hits_t` structure, so the indexing information pointing to the beginning of each cluster could change. Finally, the total energy of each cluster will most likely increase

after running through the algorithm, so this information is updated in the output information.

In summary, an `lgd_hits_t` structure and an `lgd_clusters_t` structure get passed in. They remain *unaffected* by our algorithm, which then passes out a new `lgd_hits_t` structure and a new `lgd_clusters_t` structure. Note that our algorithm will only make changes if a dead block is found to be one of the 8 blocks immediately surrounding the center block of a given cluster. Therefore, if there are no dead blocks in these locations in any of the clusters, the output structures are guaranteed to be identical to the input structures. After these new structures have been returned, it is the user's responsibility to wrap the `lgd_hits_t` structure into an `GROUP_LGD_CLUSTER_HITS` group and the `lgd_clusters_t` structure into an `GROUP_LGD_CLUSTER` group.

8.2 The Algorithm

The algorithm begins by reading in the dead block information from the dead block database. Once again, this information is based on run number and was written to the database by the algorithm of section 7.4. This allows our algorithm to know which of the LGD channels are dead.

After reading in the dead block information, the algorithm looks at each cluster one at a time. It starts by retrieving the channel number of the center block of the cluster. It then gets the channel numbers for the 8 blocks immediately bordering the center block (if the center lies along an edge, there will obviously be less than 8 immediate neighboring blocks, but this does not affect the function of the algorithm). Note that if a given block of the 8 is dead, the algorithm will necessarily be adjusting

its energy value. Be aware that these dead blocks may or may not be in the cluster already (it is possible that the block could be dead yet have a finite energy reading already, causing the clusterizer to include it in the cluster). If a dead block is not already in the cluster, this algorithm will add it to the cluster, adjusting the new C structures accordingly.

At this point, the locations of any 1st order dead blocks are known. The next step is to determine a new energy value for each of the dead blocks and to adjust the relevant data accordingly in the C structures. The simplistic algorithm that we have chosen to do this is to take an average of the energy values of every 1st order block already in the cluster that is *not* marked as dead. This energy value is then added to the energies of any dead blocks already in the cluster. If there are 1st order dead blocks not in the cluster, there energy is set exactly to this average value and the block is added into the cluster.

Once this has been done for each cluster in the `lgd_clusters_t` structure, the algorithm calculates the new total energies for each of the clusters and adjusts the data accordingly. If any new blocks were added to any clusters, the new indexing information is updated in the new `lgd_clusters_t` structure.

8.3 Additional Considerations

While the algorithm can be considered a suitable 1st order correction to the dead block problem, there are many subsequent additions which could be made to the code to make the algorithm better.

For instance, consider the method of energy extrapolation for the dead blocks.

While averaging the non-dead blocks is a good 1st approximation, it does not take into account the true center of the cluster. While we have identified the center block and the eight 1st order blocks surrounding it, the clusterizer actually refines the center of a cluster to a specific location within that center block. For instance, given a block representing the center block of a cluster, the actual recorded center of the cluster could be at the center of that block, in the upper right corner of that block, or anywhere else within the block's boundaries. Now, if we consider a case where the true center of the cluster sits at the exact center of the center block, then it is safe to assume that the incident photon struck near there and showered fairly equally among the surrounding eight 1st order blocks. In this scenario, a simple average method to determine dead block energies like the one used above should be a good method. However, if the true center of the cluster were to lie in the upper right corner of the center block, then we should assume that the 1st order blocks above and to the right of the center block should be weighted more that the 1st order blocks below and to the left. In this case, a simple average weights the lower left blocks too much and the upper right blocks not enough. Using the same logic, we could see that a dead block above and to the right of the center block should receive more energy than a block below and to the left of the center block. This would require a separate extrapolation calculation for each dead block within the cluster rather than a simple extrapolation for all of the dead blocks.

Having made this extra consideration regarding the center, another need for a better approximation can be suggested. Let us assume that the method of the previous paragraph is used and the energies of the dead blocks are based on the exact location

of the cluster. We have already stated that adding energy to the dead blocks will *move* the location of the exact center of the cluster (this is again calculated by the weighting scheme mentioned in section 5). If we were now to throw out the energies just calculated for the dead blocks and run the dead block algorithm again, the newly calculated cluster center would now weight things differently than the previous run, and we would end up with completely new energies for the dead blocks. These new energies would again shift the center of the cluster. We could theoretically continue this process of finding energies for dead blocks based on the cluster center, moving the center, and then throwing out the calculated energies and starting again. Eventually, however, it is assumed that this iteration process will eventually converge to a center that will no longer shift upon the instance of a new iteration. This iteration process would most like produce the best possible 1st order corrections.

While we have discussed only 1st order corrections, it is clear that the cluster need not be refined to include only 1st order blocks. They may include many 2nd or 3rd order blocks (2 or 3 blocks away from the center block), and any of these higher order blocks may also be dead. While it is likely that being farther away from the center, these blocks will probably have a lower energy and therefore not contribute much to the true energy and center of the cluster, 2nd or 3rd order corrections to the dead block algorithm could be proposed and implemented to improve the correction process further and produce better results.

9 RESULTS

We now present some preliminary results of these algorithms. We noted that in section 7.4, the variables in equation 7 can be adjusted to make the dead block location algorithm more conservative or more liberal. We believe that our values provide a sufficient criterion. In Figure 7, we present Figures 5 and 6 again, but this time we have placed a black line in each block that was labeled dead by the dead block algorithm. Observing this figure, we can see a few attributes worth noting. In the base test picture, there are many more blocks that appear white (assumed dead) than actually have been labeled dead. This is a sign that, as mentioned above, only using the base test as a method for locating dead blocks is not a good criterion. The LGD monitor event is clearly more suitable, for the blocks that we could assume to be dead and the labeling appear to coincide rather well. Having the base test there for comparison is still necessary, however, for there are one or two blocks that appear dead in the LGD monitor event that were actually functioning properly according to the base test.

We ran our dead block energy adjustment algorithm and analyzed all two-cluster events that were likely to be the result of a π^0 or η decay. If our algorithm was to be useful, we would expect that when the energies and other information of these clusters were collected, we would achieve a better approximation to perfect π^0 and η particle information. We found that when we only adjusted the energies of the dead blocks but did *not* recalculate the center position of the cluster, the results obtained by our algorithm were actually less accurate than those collected prior to

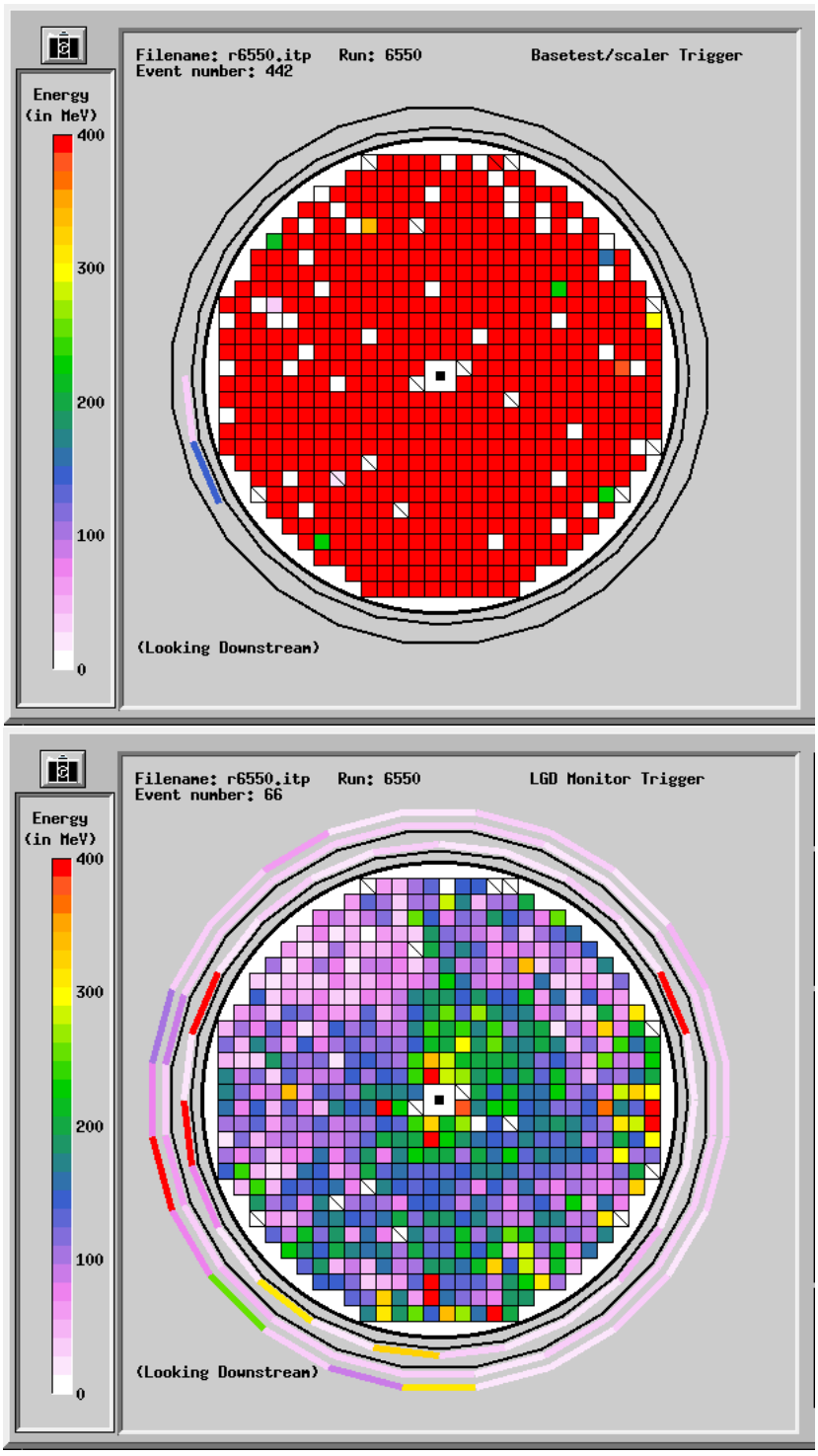

Figure 7: The LGD depiction of base test and LGD monitor events with lines through blocks labeled dead by the algorithm.

the incorporation of our algorithm. This is understandable, however, for the center of the cluster is directly connected to the momentum of the cluster. If we change the energy of the cluster without changing the center accordingly, we are adjusting the energy associated with a particle without changing its momentum. This would therefore be expected to lead to erroneous results. A second version of the algorithm in which the center of the cluster is recalculated based on the energy added to the dead blocks has been implemented at the time of this project's conclusion. However, we have not been able to gather sufficient information to deduce the success of this new adjustment. We do anticipate improvement, though. We also anticipate that if the energy added to a given dead block is based on a logarithmic scale similar to the equation used by the clusterizer to locate the center of a cluster, rather than on a simplistic linear averaging scheme, the results obtained by the algorithm will achieve a much higher level of success.

10 REFERENCES

[1]. L. Montanet et al, Review of Particle Properties, Phys. Rev. D50 (1994).

[2]. "SCALAR MESONS IN PHI RADIATIVE DECAY: THEIR IMPLICATIONS FOR SPECTROSCOPY AND FOR STUDIES OF CP VIOLATION AT PHI FACTORIES," F.E. Close, Nathan Isgur, S. Kumano, Nucl.Phys., B389, 513 (1993).

[3]. S. Okubo, Phys. Lett. 5, 1975 (1963); Phys. Rev. D16, 2336 (1977); G. Zweig, CERN Report No. 8419 TH 412, 1964 (unpublished); reprinted in Developments in the Quark Theory of Hadrons, ed. D.B. Lichtenberg and S.P. Rosen (Hadronic Press, Nonantum, MA, 1980); J. Iizuka , K. Okada, and O. Shito, Prog. Theor. Phys. Suppl. 37, 38 (1966).

[4]. "Production of Scalar K-bar K Molecules in ϕ Radiative Decays," N.N.Achasov, V.V.Gubin, V.I.Shevchenko, Phys.Rev. D56, 203 (1997).

[5]. "The Radphi Experiment at Jefferson Lab," R.T. Jones, Presented at Hadron '97 Biennial International Conference on Hadron Physics, Brookhaven National Laboratory, Upton NY, August 25, (1997).

[6]. "LIGHTEST GLUEBALL AND SCALAR MESON NONET IN PRODUCTION AND DECAY," Wolfgang Ochs (Munich, Max Planck Inst.). MPI-PHT-99-40, Jul 1999. Talk given at International Europhysics Conference on High-Energy Physics (EPS-HEP 99), Tampere, Finland, 15-21 Jul 1999.

[7]. The Radiative Phi Decay experiment Web Page,

`http://www.jlab.org/~radphi/`, May 2000.

[8]. R.A. Lindenbusch, Ph.D. thesis, Indiana University (1996).

[9]. "STUDY OF THE RADIATIVE DECAY PHI $---> $ ETA GAMMA WITH CMD-2 DETECTOR," CMD2 Collaboration, R.R. Akhmetshin et al. Phys.Lett. B460, 242 (1999).

[10]. The Event Display Web Page Web Page,

`http://www.cebaf.gov/~radphi/Software/Event.html`, May 2000.

[11]. Tom O'Connor's Senior Research Thesis, College of William and Mary, 1997 (unpublished), and The RODD Web Page,

`http://www.cebaf.gov/~radphi/Software/RODD.html`, May 2000.