ABSTRACT

Title of Thesis: Simulation and Analysis of a Hardware Trigger
for the GLUEX Experiment at Jefferson Laboratory

Name of degree candidate: James Lester Hubbard III

Degree and Year: Master of Science in Applied Physics & Computer Science 2002

Thesis directed by: David Doughty, Ph.D., Professor,
Department of Physics, Computer Science, and Engineering

GLUEX, Gluonic Excitations and Confinement, is a new experiment in the planning stages at Jefferson Lab. Part of the requirements for the experiment is the development of a hardware based level 1 trigger. The purpose of the hardware trigger is to remove background events generated by photon reactions at lower energies, while keeping the events generated by higher energy photons. The goal of this thesis is to develop a methodology to prevent background events from entering the data stream, which can be implemented within a hardware level 1 trigger.

# Simulation and Analysis of a Hardware Trigger for the GLUEX Experiment at Jefferson Laboratory

By

James L. Hubbard III

Thesis submitted to the Graduate Faculty of
Christopher Newport University in partial
fulfillment of the requirements
for the degree of
Master of Science in
Applied Physics and Computer Science
2002

Approved:

Dr. David Doughty, Chair _____

Dr. David Hibler _____

Dr. John Hardie _____

# DEDICATION

This work is dedicated to my wife, Lee Ann. Lee Ann has encouraged and supported me throughout my efforts. Her faith in my abilities has never wavered. She has been understanding of the late nights and long hours. It is also dedicated to my parents James and Donelda for being there and believing that I could achieve what I wanted. If not for the support and understanding of my loved ones none of this would be possible.

# ACKNOWLEDGMENT

This thesis would not have been possible without the help of the people who have provided direction and aid. My advisor, Dr. Dave Doughty, has been patient and understanding while I have been learning. Without his guidance, I could not have completed the work that I have. Without the help of the GLUEX collaborators my job would have been much more difficult. Dr. Elton Smith, a GLUEX collaborator, provided advice, direction, and insight when I needed it. Another, Dr. Richard Jones modified the simulation software so that I could get the information that I needed.

Ms. Lynn Sawyer and Ms. Mary Lou Anderson, also deserve credit. Ms. Sawyer has been instrumental in aiding me with the graduate school paperwork. Ms. Anderson has been patient and helpful when I have questions about general paperwork and procedures. There is no doubt that I would have still been fighting to get my assistantship pay, if she were not so persistant. If not for Dr. Cathy Roberts, I would have never known about Christopher Newport University. She convinced me that it had a good program, and I have not been disappointed.

I would also like to thank the Physics, Computer Science, and Engineering Department and faculty. The PCSE Department truly is outstanding. The quality of teaching and the accessibility of the faculty are superb.

# TABLE OF CONTENTS

# LIST OF FIGURES

**Number** **Page**

# LIST OF TABLES

**Number** **Page**

# Chapter 1

# Introduction

GLUEX, Gluonic Excitations and Confinement, is a new experiment project in the planning stages at Jefferson Lab. The purpose of GLUEX is to "study the photoproduction of unusual mesons and gluonic excitations"[1, pp. 1]. In order to study these, a beam of high energy photons, a detector and a target are needed. The photoproduction occurs when photons, with energy of 9 $GeV$, strike a hydrogen atom. This reaction produces other nuclear particles that propagate through the detector, which records the interaction of the particles with the various detector packages.

## 1.1 Photon Beam

While there are a couple of different techniques for producing a photon beam, the method best suited for the facilities of Jefferson Lab is coherent bremsstrahlung radiation. Photons are generated by placing a thin diamond wafer radiator in the electron beam. By far, the problem that has the greatest impact on the design of the level 1 hardware trigger is the energy spectrum of the photon beam.

When oriented a certain way the radiator yields a photon energy spectrum which peaks at 9 $GeV$. Unfortunately, a large part of the beam spectrum will be below the desired peak as shown in Fig. 1.1[1, pp. 55]. The combination of the energy spectrum and the cross

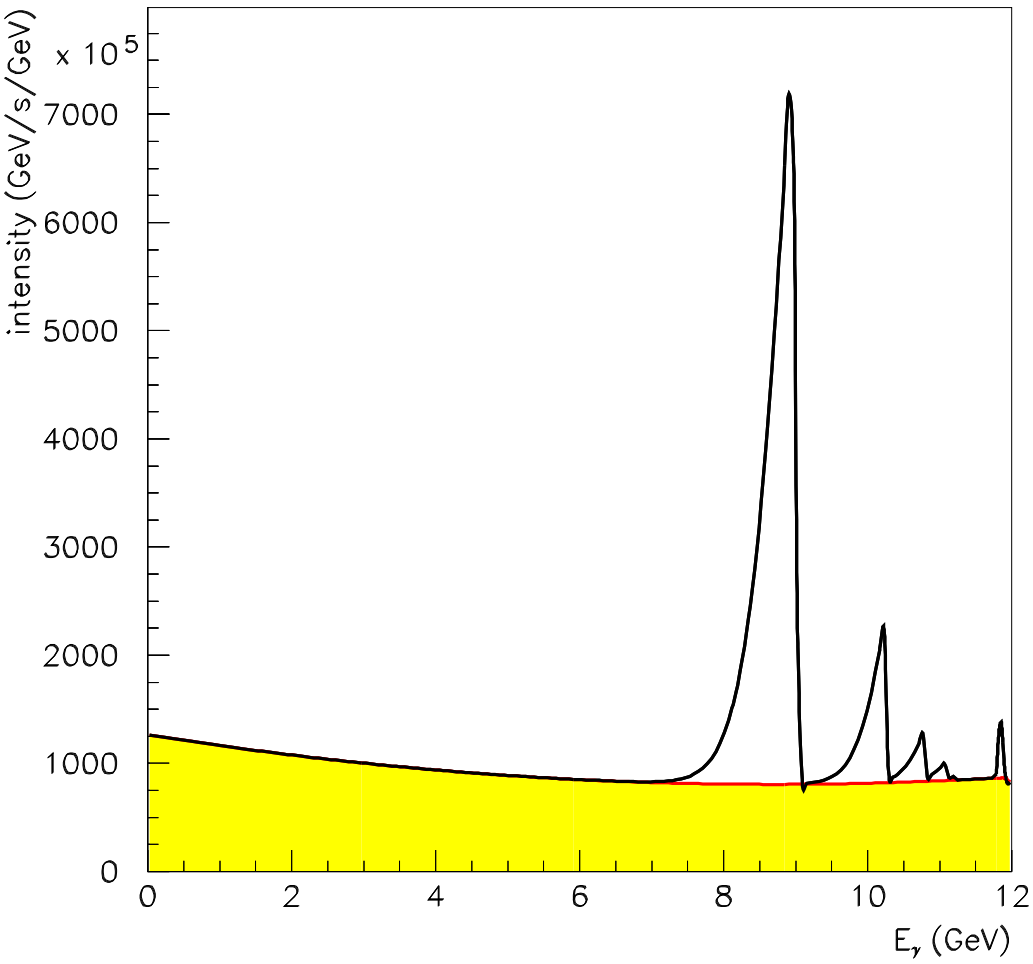


Figure 1.1: Photon energy spectrum from an oriented diamond radiator. The $y$ axis is $dP/dE$ with energy $P$ expressed in $GeV$/s and $E$ in $GeV$. The radiator thickness is $10^{-4}$ radiation lengths and the electron beam current is $1\ \mu A$. Shown is what emerges after the photon beam passes through a collimator $3.4\,mm$ in diameter located $80\ m$ downstream from the radiator. [1, pp. 55]

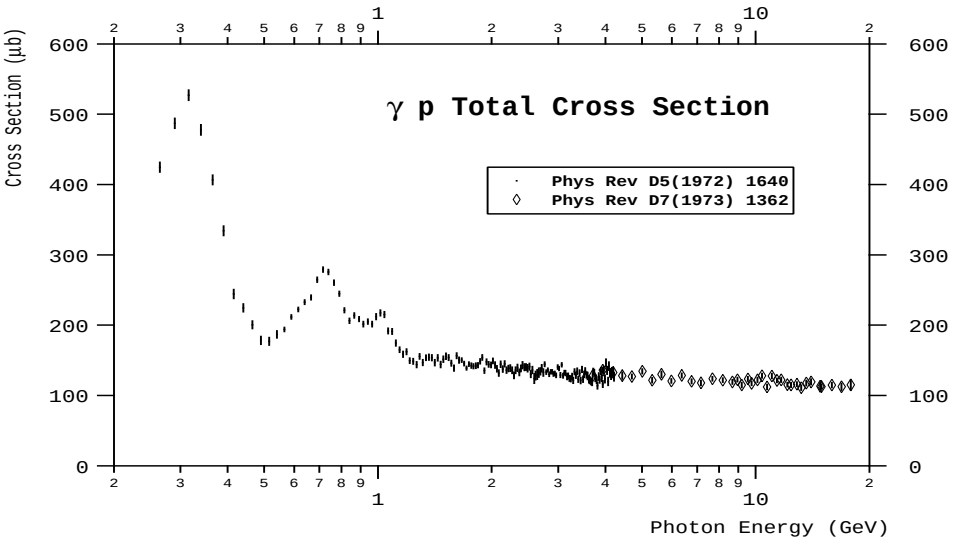


Figure 1.2: Total cross section for $\gamma p \rightarrow$ *hadrons* as a function of photon energy.[1, pp. 208]

section, shown in Fig. 1.2[1, pp. 208], demonstrates that a large number of the reactions will be occurring outside of the experimentally desirable range. These unwanted reactions will contribute a great deal of background events to the experiment. At a tagged rate of $10^7$ photons/s, the tagged hadronic rate is expected to be 1.4 *kHz*, while the contributions of the background events create a total hadronic rate of 37 *kHz*. When the tagged rate is $10^8$ photons/s, the expected rates increase by an order of magnitude. The tagged hadronic rate increases to 14 *kHz*, while the total hadronic rate increases to 385 *kHz*. [1, ch. 8]

## 1.2 Detector

The GLUEX detector's major subsystems are shown in Fig. 1.3 [1, pp. 122]. Photons strike the liquid hydrogen target causing a reaction which produces additional particles. Energy measurements of the reaction products are taken using the barrel calorimeter and the lead glass array, also known as the forward calorimeter. Tracking information is provided by the vertex chamber/start counter and both the central and forward drift chambers. The forward time-of-flight (TOF), though not shown, is connected to the forward calorimeter. Together,

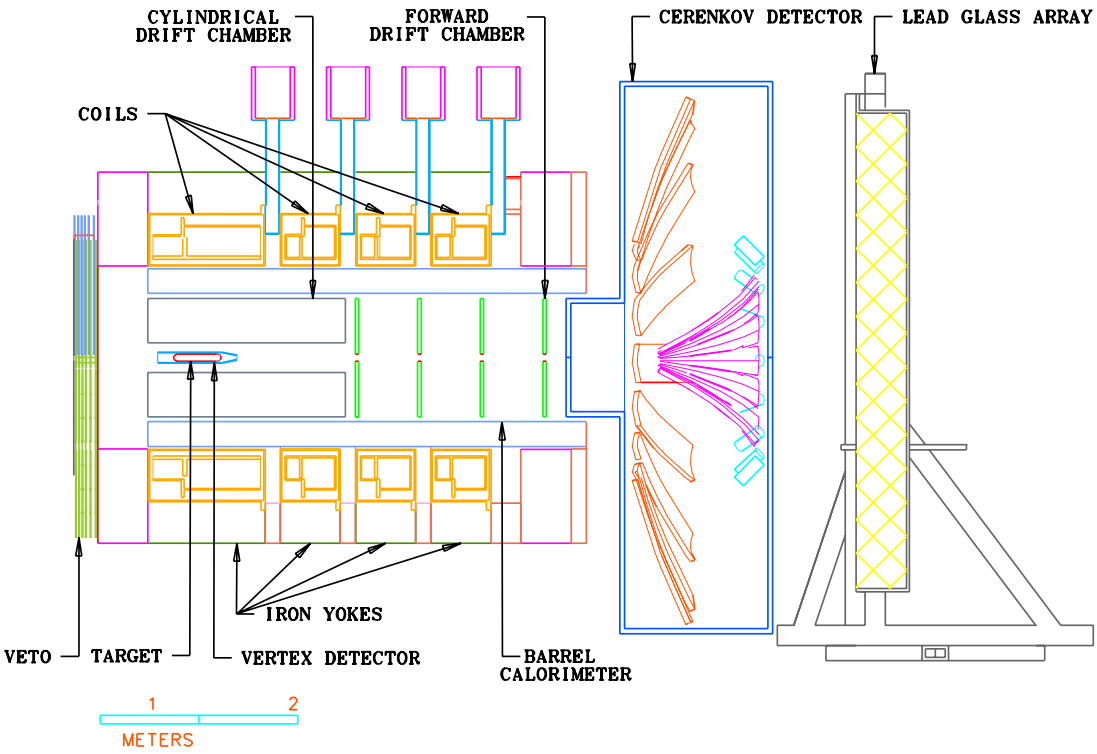


Figure 1.3: An overview of the GLUEX detector. The major subsystems are labeled. [1, pp. 55]

the cerenkov detector and the forward TOF will perform identification of charged particles [1, pp. 174]. In addition to particle identification, our expectation is that the forward TOF can provide rudimentary information about hits or "tracks".

Because the trigger must operate very quickly, the most useful subsystems are the ones that have the lowest latency. The start counter, barrel calorimeter, forward calorimeter and forward TOF fulfill this requirement. The drift chambers are not used because there is a high latency between charged particles traveling through a chamber, and the signal that its passing causes.

Data taken from calorimeters will be used to compute an energy sum for each calorimeter. The start counter can provide a total number of tracks produced by a reaction. Additional information about track counts will be provided by the forward TOF.

## 1.3 Triggering

The high event rates caused by the large number of background events presents a challenge for the trigger. At $10^7$ photons/s and an event rate of 37 *kHz*, the system is generating 180 MB/s of data, if each event is 5 KB. At $10^8$ photons/s and an event rate of 385 *kHz*, the amount of data generated is 1.76 GB/s. Unfortunately, much of the data being generated is not interesting because of the large number of background events being captured.

To maximize the amount of good data entering the data acquisition (DAQ) system while minimizing the background produced by the accidentals, a triggering system needs to be developed. For the GLUEX experiment two triggering systems are planned. The first is hardware based and is known as the level 1 hardware trigger. The second is called the level 3 software trigger.

The level 1 hardware trigger's job is to prevent as many of the accidentals from entering the DAQ system as possible. This is accomplished by analyzing information from the detector subsystems, which can provide data very quickly. The trigger performs a fast analysis of the data using a set of criteria or a function to determine if an event is a background or good event.

The level 3 software trigger will run on a cluster of machines and analyzes the data after it has been captured by the DAQ. This trigger analyzes data from all detector packages, including the tracking chambers, to find good events. After finishing the analysis and removing any background events, the data is sent to storage.

Combined, the goal of both the level 1 and 3 triggers is to reduce the total rate to the acutal tagged hadronic rate. Going from an event rate of 37 *kHz* to 1.4 *kHz* tagged equates to a reduction of 20:1. With an event rate of 1.4 *kHz* and event block data amount of 5 KB, the data rate to storage is 6.8 MB/s. When the tagged rate is 14 *kHz*, the data rate is 68 MB/s. These data rates are much easier to manage.

The focus of this thesis is to develop an algorithm that is usable by the level 1 trigger to prevent background events from entering the data stream. The tentative goal of the leve1 1

trigger is to remove at least 50% of the background events in real time without removing more than 0.5% of the desirable events. Two techniques for cutting have been examined. The first method uses a series of conditional tests. The second technique, which has proven itself the most capable, uses a computed function to decide if an event is a background event.

# Chapter 2

# Methodology

As stated previously, GLUEX is in the planning stages, which means that the detector and beam are unavailable. Most of the major subsystems are in the design phase now, except the forward calorimeter, which had been used in another experiment. The designers are building on past knowledge and experience. This knowledge and experience is used to construct simulations.

Simulations are necessary for design. Event information is generated based on previous experimental results, and is used in the detector simulation. A detector simulation uses information about particle interactions to pass particles through detector subsystems. As these particles pass through the detector, the simulation produces data about the interactions. This data can be analyzed, and from the analysis design decisions can be made that will allow a working prototype to be built. The prototype serves as a basis for doing preliminary experiments, which allows the simulation to be modified to produce better results.

To design the level 1 hardware trigger, we relied heavily on the software that the GLUEX collaborators have already built. The primary tools they supplied were the event generator and detector simulation. Using these tools we were able to simulate various reactions and perform an analysis of the performance of many trigger designs.

## 2.1 Reactions

For simulation and analysis the six reactions in Table 2.1 were chosen. They are representative of reactions that are of interest (such as the $\gamma p \rightarrow \rho^0\pi^+ n \rightarrow n\pi^+\pi^-\pi^+$), or are background. The two $\Delta$ reactions are purely background and only occur at the low end of the photon energy spectrum. The other four are more representative of reactions that will occur at peak energy levels. However, they too can occur at lower energy levels. Another reason for choosing this set of reactions was to obtain a mix of event types, with some having a lot of neutral energy, while others have more charged particles.

| Reaction | Energy($GeV$) | Type |
|---|---|---|
| $\gamma p \rightarrow \rho^0\pi^+ n \rightarrow n\pi^+\pi^-\pi^+$ | 1 | Bkgrd |
| | 2 | Bkgrd |
| | 9 | Good |
| $\gamma p \rightarrow \rho p \rightarrow p\pi^+\pi^-$ | 1 | Bkgrd |
| | 2 | Bkgrd |
| | 9 | Bkgrd |
| $\gamma p \rightarrow X^*(1600)n \rightarrow (\eta^0\pi^+)n \rightarrow n\pi^+\gamma\gamma$ | 1 | Bkgrd |
| | 2 | Bkgrd |
| | 9 | Bkgrd |
| $\gamma p \rightarrow X^+(1600)\Delta^0 \rightarrow (\pi^+\pi^+\pi^-)(n\pi^0) \rightarrow \pi^+\pi^+\pi^- n\gamma\gamma$ | 9 | Good |
| $\gamma p \rightarrow \Delta \rightarrow n\pi^+$ | 0.337 | Bkgrd |
| $\gamma p \rightarrow \Delta \rightarrow p\pi^0$ | 0.337 | Bkgrd |

Table 2.1: These reactions were chosen for simulation and analysis. Three of the reactions were also simulated at 1 $GeV$ and 2 $GeV$ to represent interesting reactions occurring at energy levels where they would be considered background. The two $\Delta$ reactions are background only.

## 2.2 Simulation Tools

The software packages needed to perform the simulations have already been built by the GLUEX collaboration. Generating data which can be used for analysis requires the series of steps shown in Fig 2.1. Simulation begins by generating the reaction products of a photon-

proton collision, The reaction description file details how the products of the reaction are created. Genr8 [2] is a program that produces an event file containing a specified number of events based on the event description supplied by the reaction description file. Each event specifies the particle's energy, momentum, and the direction of all particles in the event.

Before the events can be used in the simulation the format of the data file must converted to the Hall D Data Model or hddm [3]. Hddm is a modified form of Extensible Markup Language (XML). XML is a self documenting method of formatting data which uses tags to describe the data. The hddm format is a stream of data that is in a form similar to XML, except the XML tags have been stripped from the data stream to conserve space. The XML description included at the beginning of the hddm file can be used to convert the format into a true XML file.

The event information from Genr8 is converted to hddm by using two other conversion programs. After conversion the hddm file produced is run through the HDGeant [4] Monte Carlo simulation package. The detector geometry and other information about the detector packages are also used by the simulation. By modifying these files and rebuilding HDGeant, the simulation can be changed. Using the event information file and the detector information files, HDGeant models the behavior of the detector packages as each particle passes through. The end result is a file that contains information about how each particle, in each event, interacted with the detector during the simulation.

These simulation tools can be compiled and run on any system that supports the standard GNU compilers. Much of the code is in C, but there is some FORTRAN and shell scripting involved. Current Linux distributions can be used to build and run these tools.

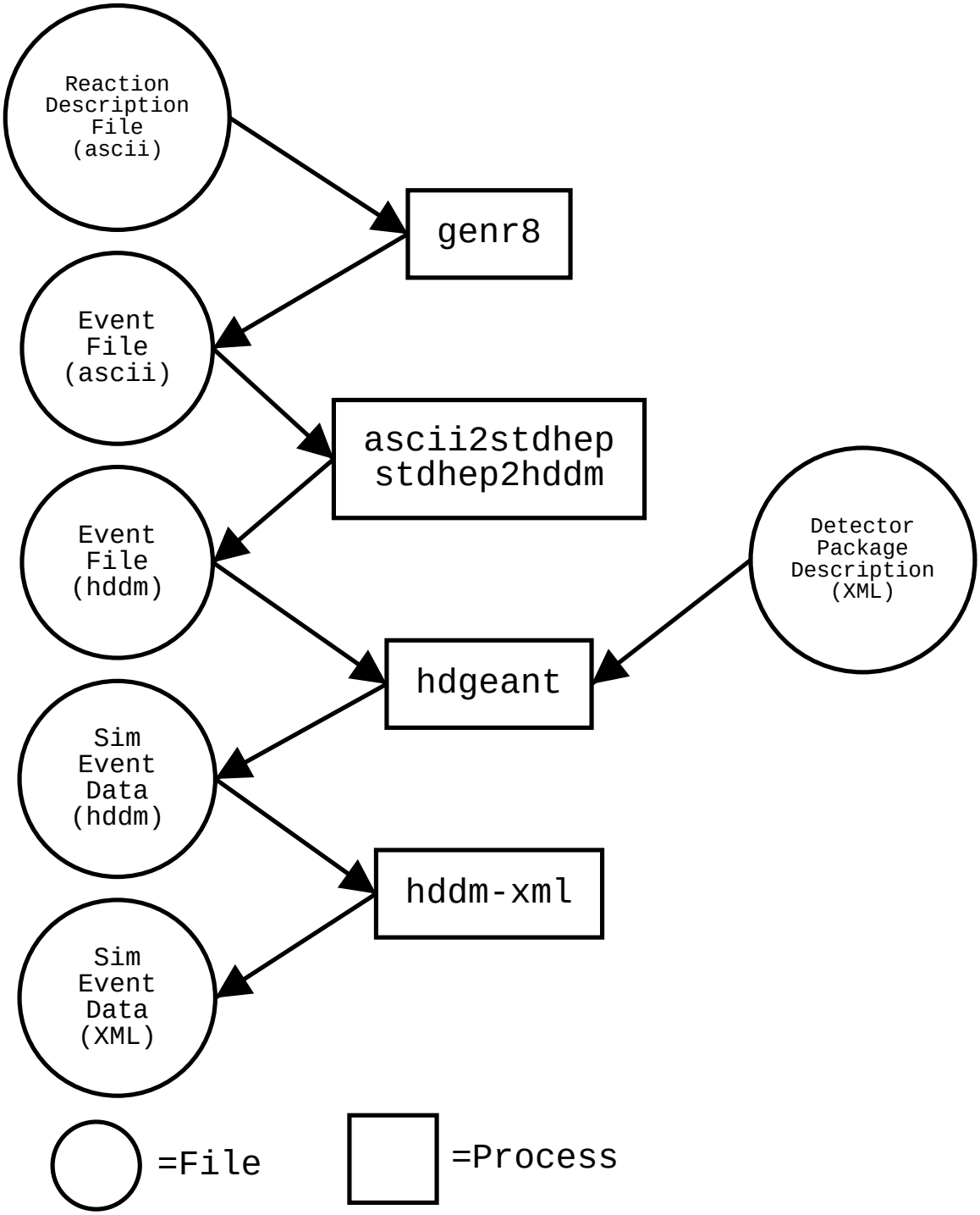


Figure 2.1: This diagram shows the chain of events required to go from an initial reaction description file to the final simulation data file in xml format.

## 2.3 Analysis Tools

After simulating the reactions, the data needs to be converted to another format for use by the analysis tools. We chose to convert the data from hddm into XML form. XML was chosen because a converter was available, XML is self documenting, and there are a plethora of tools available to manipulate the files. To enable the analysis tools to read the XML data files, Sun's Java Architecture for XML Binding (JAXB) [5] was used.

JAXB is a new API and specification from Sun. JAXB's primary purpose is to provide a way of automatically mapping between XML and Java objects. To access data in XML format, a Document Type Definition (DTD) and a schema binding representation are needed. Together, these files are used by a binding compiler to generate the class files with methods to move data between Java object representation and XML format.

Part of a simulation data file is shown below. The data for an event starts with the *physicsEvent* tag. Data about the event is stored in the attribute. *runNo* stores data about what simulation produced the event. *eventNo* is the number of the event. The *product* tag describes the products of the reaction. The first particle produced was the pi+. Information about the particle is stored in the attributes of the *momentum* tag.

n3pi-1gev

```
<physicsEvent runNo="-9000" eventNo="1">
  <reaction type="0" weight="0">
    <vertex>
      <product type="pi+" decayVertex="0">
        <momentum E="0.264355" px="-0.073488" py="-0.077761"
                  pz="0.197069" />
        <properties mass="0.14" charge="1" />
      </product>
```

The following text is part of the DTD file, detailing how the XML parameters arrive. *physicsEvent* is an element that defines an XML tag with two child tags called *reaction* and *hitView*. Since *reaction* occurs first, it must be the next XML tag after *physicsEvent*. The

attributes of *physicsEvent* are shown next. These are located within the tag and have data associated with them.

halldxml.dtd

```
<!ELEMENT physicsEvent (reaction, hitView) >
                <!ATTLIST physicsEvent
                          runNo CDATA #REQUIRED
                          eventNo CDATA #REQUIRED >
        <!ELEMENT reaction (vertex) >
                <!ATTLIST reaction
                          type  CDATA #REQUIRED
                          weight CDATA #REQUIRED >
```

The next section contains part of the schema binding from the *halldxml.xjs*, which maps the data in the XML file to actual data types. As can be seen the XML tag names are mapped to classes and the attributes are given a data type.

halldxml.xjs

```
<element name="physicsEvent" type="class" root="true" >
      <attribute name="runNo" convert="int"/>
      <attribute name="eventNo" convert="int"/>
</element>
<element name="reaction" type="class">
      <attribute name="type" />
      <attribute name="weight" convert="float"/>
</element>
```

The code section shown below is part of the *PhysicsEvent.java* source code file. It was generated by the JAXB binding compiler based on the information from *halldxml.xjs* and *halldxml.dtd*. The XML tags and attributes from the DTD are clearly visible as data types. A few of the methods that are available to the programmer to view and modify data are visible as well.

PhysicsEvent.java

```
public class PhysicsEvent
```

```
    extends MarshallableRootElement
    implements RootElement
{

    private int _RunNo;
    private boolean has_RunNo;
    private int _EventNo;
    private boolean has_EventNo;
    private Reaction _Reaction;
    private HitView _HitView;

    public int getRunNo() {
        if (has_RunNo) {
            return _RunNo;
        }
        throw new NoValueException("runNo");
    }

    public void setRunNo(int _RunNo) {
        this._RunNo = _RunNo;
        has_RunNo = true;
        invalidate();
    }

    public boolean hasRunNo() {
        return has_RunNo;
    }
```

Physics Analysis Workstation (PAW)[6] is a well known tool that has been extensively used by nuclear and particle physicists to perform data analysis and display results. However, we chose to use Java Analysis Studio [7] (JAS) as our primary analysis tool. JAS is an application that is targeted toward users who need to create histograms and other plots for data analysis. Plots are created by writing Java code to access data and fill in histograms. One of JAS's strengths is its ability to read many of the standard physics data formats. The deciding factors in choosing JAS were the ease in extending it to read the XML files generated by the simulation, and the ability to use Java code in the analysis. To add the capability to read these XML files, a Data Interface Module (DIM) was needed. The Hall D XML DIM was written in Java and used JAXB to create the interfaces needed by JAS to read the data. This meant that everything needed to do analysis could be performed within

the Java environment and could be run on any machine with a Java Virtual Machine.

# Chapter 3

# Analysis

The hardware level one trigger has to analyze data very quickly from the detector so that it can prevent background events from entering the data stream. To do this, it can only use data from those subsystems that output the data very quickly. This limits the useful data to the energy in the forward and barrel calorimeters, and the number of tracks in both the start counter and the forward TOF. Traditionally, a trigger system operates with a group of conditional statements which test for certain conditions. Using these tests, the level 1 trigger makes a decision about the event and signals the DAQ system to either throw away or read out the data. When the DAQ system reads out the data, the data is passed on to the level 3 trigger.

## 3.1 Conditional Form

Initially, our research focused on using multiple conditional statements to cut the background events. Using JAS, various event data were studied. As can be seen in Fig. 3.1, the amount of energy in both the calorimeters is almost always greater than 0.5 $GeV$ when the photon energy is at 9 $GeV$. On the other hand, when the interacting photon is around 1 $GeV$, there are many events where the energy is less than 0.5 $GeV$, see Fig. 3.2. Also of note is that most of the event's energy is in the forward calorimeter when photon energy is

9 $GeV$. When photon energy is 1 $GeV$, much of the energy is in the barrel calorimeter.

Another feature that we relied on was using the track counts in the forward TOF. When the photon energy is 9 $GeV$, the forward TOF almost always has a particle track as shown in Fig. 3.3. As can be seen in Fig 3.4, at 1 $GeV$ there are close to 1300 events where there were no particles in the forward TOF.

Using the above information, we experimented with different conditional statements to develop an optimized algorithm that would cut background events from the data stream. To rank the efficiency of various triggering algorithms, an evaluation function was used. The evaluation function is shown in Eqn. 3.1. This function produces a value between zero and one, where a result of zero means the trigger did not cut any background events and cut all of the good events. A result of one means the trigger was perfect and cut all of the background events and kept all of the good events.

| |
|---|
| Total Calorimeter Energy $< 0.5\ GeV$<br>AND<br>Forward Calorimeter Energy $<$ Barrel Calorimeter Energy |
| OR |
| Forward Calorimeter Energy $< 0.5\ GeV$<br>AND<br>Forward TOF Track Count = 0 |

Table 3.1: Conditional tests to cut background events.

The conditional statements shown in Table 3.1, provided the best results. The trigger meets the goal of cutting no more than 0.5% of the high energy events. Applying the evaluation function produced a score of 0.786. As Table 3.2 shows, it was still not cutting as many of the low energy events as we would like. We thought it should be possible to get better results.

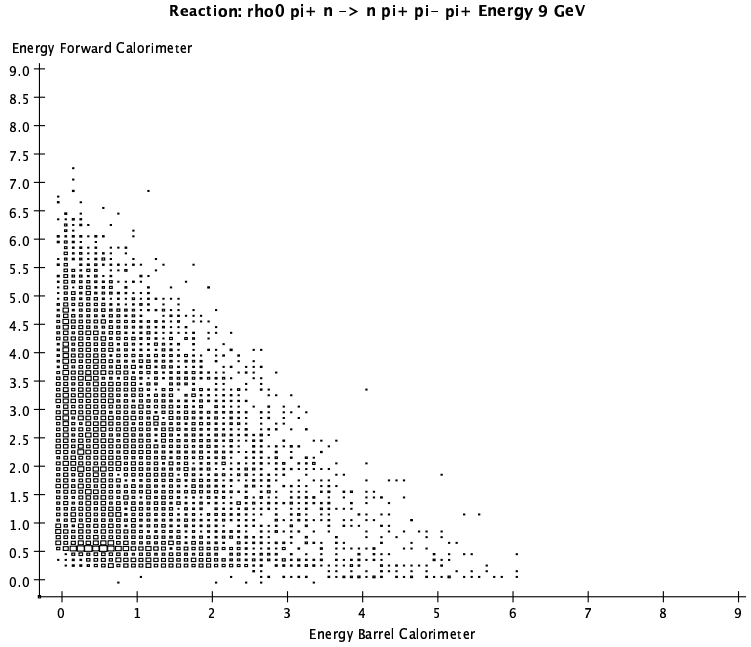


Figure 3.1: Energy in calorimeters for $\gamma p \to \rho^0 \pi^+ n \to n\pi^+\pi^-\pi^+$ at 9 $GeV$. The $y$ axis is the amount of energy in the forward calorimeter. The $x$ axis is the amount of energy in the barrel calorimeter. Both are in $GeV$.

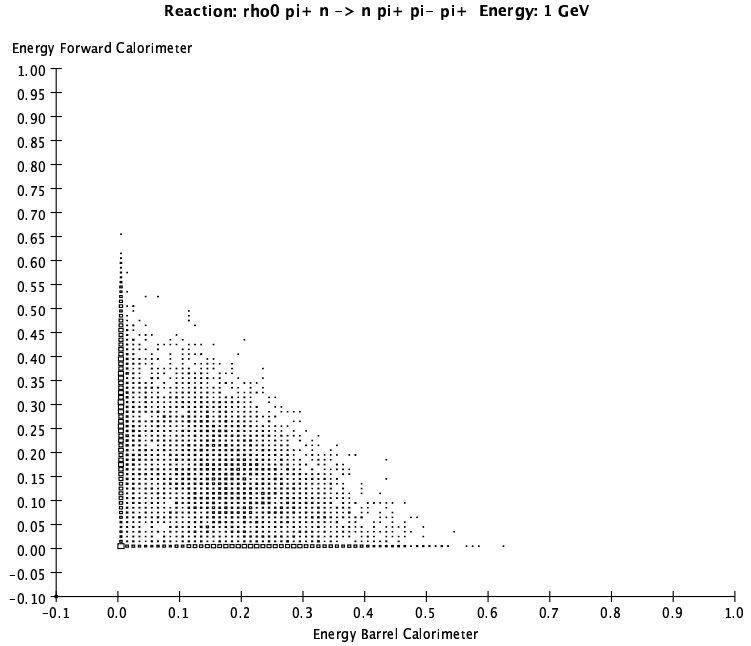


Figure 3.2: Energy in calorimeters for $\gamma p \to \rho^0 \pi^+ n \to n\pi^+\pi^-\pi^+$ at 1 $GeV$. The $y$ axis is the amount of energy in the forward calorimeter. The $x$ axis is the amount of energy in the barrel calorimeter. Both are in $GeV$.

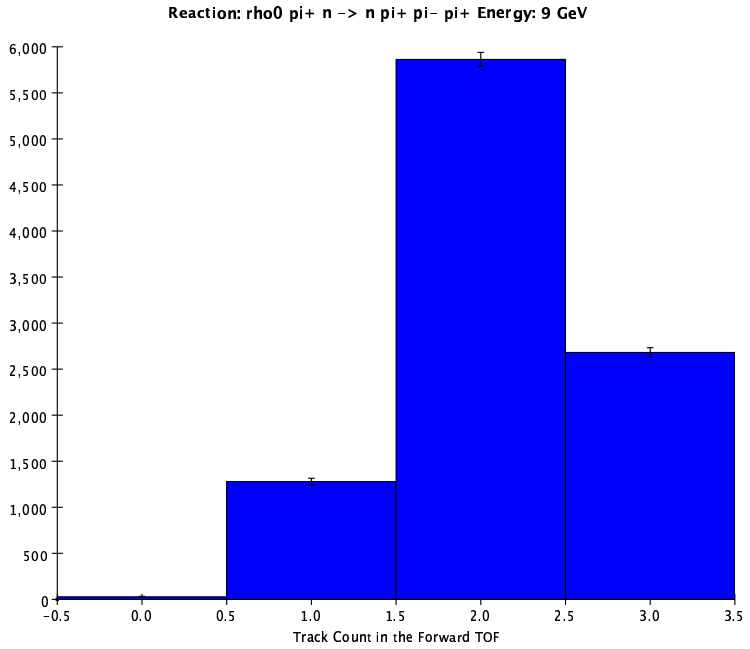


Figure 3.3: Track Counts in the Forward TOF for $\gamma p \rightarrow \rho^0\pi^+n \rightarrow n\pi^+\pi^-\pi^+$ at 9 $GeV$. The $y$ is the number of events. The $x$ axis is the number of track counts in the forward TOF. A zero means that none of the event's particles hit the TOF.

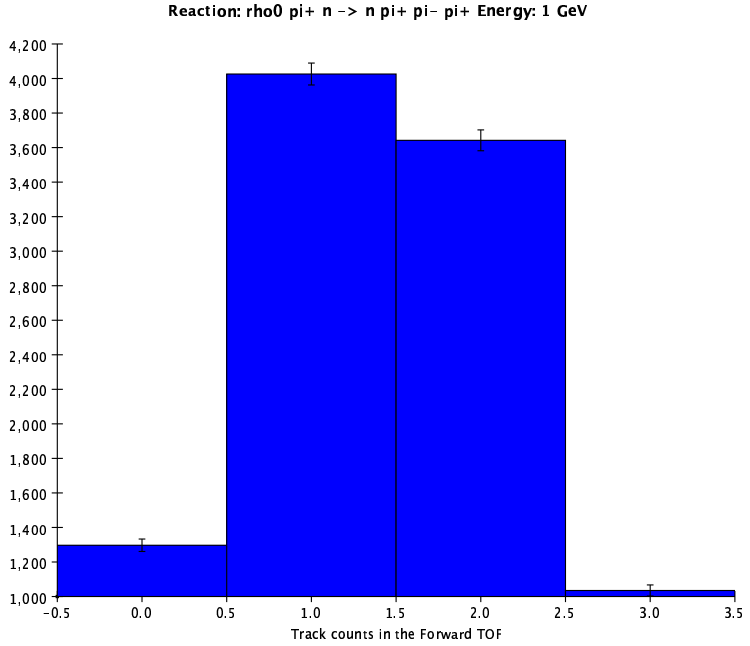


Figure 3.4: Track Counts in the Forward TOF for $\gamma p \rightarrow \rho^0\pi^+n \rightarrow n\pi^+\pi^-\pi^+$ at 1 $GeV$. The $y$ is the number of events. The $x$ axis is the number of track counts in the forward TOF. A zero means that none of the event's particles hit the TOF.

| Reaction | Energy($GeV$) | Percent Cut |
|---|---|---|
| $\gamma p \to \rho^0 \pi^+ n \to n\pi^+\pi^-\pi^+$ | 1 | 45.07% |
| | 2 | 30.88% |
| | 9 | 0.25% |
| $\gamma p \to \rho p \to p\pi^+\pi^-$ | 1 | 61.06% |
| | 2 | 47.18% |
| | 9 | 0.30% |
| $\gamma p \to X^*(1600)n \to (\eta^0\pi^+)n \to n\pi^+\gamma\gamma$ | 1 | 42.29% |
| | 2 | 53.89% |
| | 9 | 0.04% |
| $\gamma p \to X^+(1600)\Delta^0 \to (\pi^+\pi^+\pi^-)(n\pi^0) \to \pi^+\pi^+\pi^- n\gamma\gamma$ | 9 | 0.5% |
| $\gamma p \to \Delta \to n\pi^+$ | 0.337 | 81.99% |
| $\gamma p \to \Delta \to p\pi^0$ | 0.337 | 97.73% |
| Evaluated Score 0.786 | | |

Table 3.2: Conditional trigger cut rates for reactions and their energies.

$$F = \quad ( [NumberOfBackgroundReactions] * \frac{[TotalNumberOfBackgroundEventsCut]}{[TotalNumberOfBackgroundEvents]} + 2 * [NumberOfDesiredReactions] * \frac{[TotalNumberOfDesiredEventsNotCut]}{[TotalNumberOfDesiredEvents]} )/ ( [NumberOfBackgroundReactions] + 2 * [TotalNumberOfDesiredEvents] ) \tag{3.1}$$

## 3.2 Functional Form

During a discussion of the conditional trigger results with fellow GLUEX collaborators, one individual suggested using a general function[8]. Using a general functional form, we could divide the multi-parameter space with a hyper-surface that could provide a much finer tuning than the straight planes that the conditional triggers were providing. We began looking into various functional forms and methods of function optimization.

### 3.2.1 Function

Building on our earlier efforts to find a group of conditional statements to perform cuts, a function was developed in the form of Eqn. 3.2. This function uses the same variables as the conditional trigger and four coefficients. Computing the equation results in an answer which is compared to $Z$. If the result is less than or equal to $Z$, the event is cut.

$$\begin{aligned} Z \;>=\; & A * [NumberTracksForwardTOF] \\ + \; & B * [EnergyForwardCal] \\ + \; & C * \frac{[EnergyForwardCal + 1]}{[EnergyBarrelCal + 1]} \end{aligned} \tag{3.2}$$

### 3.2.2 Optimization

Having a function with coefficients presents the problem of determining the values that the coefficients should take. This is an optimization problem and there are several well known techniques for performing optimizations. Among various methods, neural networks are one of the better known techniques for learning. They use a gradient descent method that allows them to learn patterns. The down side is that they can be slow, hard to train and can easily get trapped at local minima.

Another method for doing optimization is to use genetic algorithms. Genetic algo-

rithms randomly search a solution space. One result of this is that they do not fall into the trap of local minima or maxima. If they do, it is relatively easy to modify the algorithm to avoid these traps. Another plus is that, because of their randomness, they are fast in searching a solution space. An additional advantage is that there are many freely available implementations that make it relatively easy to use genetic algorithms.

The downside to using a genetic algorithm is that it may have to be run multiple times to ensure that a "good enough" solution has been found. This is because it randomly walks the search space in a finite amount of time. We chose to use genetic algorithms based on the strong points, and minimized the problem using many repetitions.

Genetic algorithms are modeled after the basic ideas of evolution. The chromosome represents a possible solution to the problem. A group of chromosomes make up a population. By using crossover and mutation operations, the population of solutions change each generation. During each evolution of the population, each solution is evaluated using a fitness function. The score that it receives, combined with the selection operator being used, determines if the chromosome will be used in the next generation.

The fitness function plays a crucial role in finding a solution. While the crossover, mutation, and selection operators can be changed, the fitness function is chosen entirely by the user. It scores the chromosome based on some criteria or function. After assigning it a score, the selection process determines which chromosomes continue into the next generation. When it has finished evolving the solution for a number of generations, the solution with the highest score is chosen.

Our fitness function scored the chromosomes on how well the solution they represented cut background events while preserving the good events. In order to do this, the score of the chromosome would be "punished" by subtracting points when it would cut the desirable events. When doing the right thing, score would be increased. The criteria for our fitness function is listed below.

- +1 point for cutting a background event

- +5 points for not cutting a good event
- -50 points for cutting a good event
- reset score to zero for cutting more than 50 good events.

# Chapter 4

# Results and Conclusions

To find the solution that performed the best, we used the evaluation function discussed previously. After running the optimizer several hundred times, the solutions were ranked. The best solution for the coefficients of the cut function was equation 4.1. This function cut more events than the conditional trigger by a large margin. The conditional trigger had a score of 0.786, while the functional trigger had a score of 0.861. The results of the functional trigger can be seen in the Table 4.1. The functional trigger met the goal of cutting no more than 0.5% of the good events, but it also surpassed the goal of cutting 50% of the background events by cutting an average of 72% of the background events.

$$\begin{aligned} -0.520 \;\; >= \;\; & -0.168 * [NumberTracksForwardTOF] \\ + \;\; & 7.972 * [EnergyForwardCal] \\ + \;\; & -1.855 * \frac{[EnergyForwardCal + 1]}{[EnergyBarrelCal + 1]} \end{aligned} \tag{4.1}$$

Even though the results were good, we were concerned that the simulation might not be modeling the hadronic energy deposition in the forward calorimeter properly. In order to deal with this, the hadronic energy deposition was varied by +20% and -20% using the

| Reaction | Energy($GeV$) | Percent Cut |
|---|---|---|
| $\gamma p \rightarrow \rho^0 \pi^+ n \rightarrow n\pi^+\pi^-\pi^+$ | 1 | 67.99% |
| | 2 | 41.68% |
| | 9 | 0.05% |
| $\gamma p \rightarrow \rho p \rightarrow p\pi^+\pi^-$ | 1 | 70.48% |
| | 2 | 54.82% |
| | 9 | 0.50% |
| $\gamma p \rightarrow X^*(1600)n \rightarrow (\eta^0\pi^+)n \rightarrow n\pi^+\gamma\gamma$ | 1 | 90.10% |
| | 2 | 56.24% |
| | 9 | 0.11% |
| $\gamma p \rightarrow X^+(1600)\Delta^0 \rightarrow (\pi^+\pi^+\pi^-)(n\pi^0) \rightarrow \pi^+\pi^+\pi^- n\gamma\gamma$ | 9 | 0.23% |
| $\gamma p \rightarrow \Delta \rightarrow n\pi^+$ | 0.337 | 99.99% |
| $\gamma p \rightarrow \Delta \rightarrow p\pi^0$ | 0.337 | 98.75% |
| Evaluated Score 0.861 | | |

Table 4.1: Functional trigger cut rates for reactions and their energies.

existing data sets. The analysis was re-run against the modified data sets. The results were comparable to the results for the original trigger. When the hadronic energy was increased by +20%, the score of the highest solution was 0.861. When it was decreased by -20%, the score became 0.863. The results for both data sets can be seen in Tables 4.2 and 4.3.

## 4.1 Conclusion

The results that have been obtained are promising. They show that a level 1 hardware trigger can be built which will meet the goal of cutting greater than 50% of the background events while keeping the good events. While the simulation may not be perfect, the results of modifying energy deposition show that our method is adaptable. The functional trigger will play a major role in cutting down the data stream to a size that is manageable by the data acquisition system.

While this thesis has been a proof of concept, it lays down the ground work necessary to implement a more robust analysis system. This system will be necessary in order to analyze the data coming from a real detector in the initial stages of testing and running.

| Reaction | Energy($GeV$) | Percent Cut |
|---|---|---|
| $\gamma p \to \rho^0 \pi^+ n \to n\pi^+\pi^-\pi^+$ | 1 | 66.41% |
| | 2 | 40.7% |
| | 9 | 0.08% |
| $\gamma p \to \rho p \to p\pi^+\pi^-$ | 1 | 70.64% |
| | 2 | 54.91% |
| | 9 | 0.50% |
| $\gamma p \to X^*(1600)n \to (\eta^0\pi^+)n \to n\pi^+\gamma\gamma$ | 1 | 90.71% |
| | 2 | 57.12% |
| | 9 | 0.10% |
| $\gamma p \to X^+(1600)\Delta^0 \to (\pi^+\pi^+\pi^-)(n\pi^0) \to \pi^+\pi^+\pi^- n\gamma\gamma$ | 9 | 0.22% |
| $\gamma p \to \Delta \to n\pi^+$ | 0.337 | 99.99% |
| $\gamma p \to \Delta \to p\pi^0$ | 0.337 | 99.08% |
| Evaluated Score 0.861 | | |

Table 4.2: Functional trigger cut rates for reactions and their energies. The hadronic energy deposition has been increased by 20% in the forward calorimeter.

| Reaction | Energy($GeV$) | Percent Cut |
|---|---|---|
| $\gamma p \to \rho^0 \pi^+ n \to n\pi^+\pi^-\pi^+$ | 1 | 65.72% |
| | 2 | 43.20% |
| | 9 | 0.13% |
| $\gamma p \to \rho p \to p\pi^+\pi^-$ | 1 | 73.42% |
| | 2 | 56.03% |
| | 9 | 0.50% |
| $\gamma p \to X^*(1600)n \to (\eta^0\pi^+)n \to n\pi^+\gamma\gamma$ | 1 | 89.43% |
| | 2 | 55.61% |
| | 9 | 0.10% |
| $\gamma p \to X^+(1600)\Delta^0 \to (\pi^+\pi^+\pi^-)(n\pi^0) \to \pi^+\pi^+\pi^- n\gamma\gamma$ | 9 | 0.22% |
| $\gamma p \to \Delta \to n\pi^+$ | 0.337 | 99.99% |
| $\gamma p \to \Delta \to p\pi^0$ | 0.337 | 99.41% |
| Evaluated Score 0.863 | | |

Table 4.3: Functional trigger cut rates for reactions and their energies. The hadronic energy deposition has been decreased by 20% in the forward calorimeter.

# Appendix A

# Genetic Algorithm

## A.1 Concepts

Genetic algorithms are an evolutionary programming technique that borrows some basic ideas from biological science. Although it may be an oversimplification, evolution theory basically states that individuals that are most adaptable to their environment will survive and pass on their genetic adaptations to their offspring. This in turn should make their offspring even more adapted to their environment.

Genetic algorithms (GA) make use of a simplified version of genes, chromosomes, crossover, mutation, and fitness selection. Using these operators, a problem can be coded that will search a solution space.

The most simple chromosome can be considered an array of integers. Each location in the array would be considered a gene. Each gene is part of the solution for the problem. A population of chromosomes have to be initialized. Typically they are randomly initialized so that each time the GA is run it can possibly access a new part of the search space.

To allow an GA to find better solutions, crossover and mutation are used to create children that may be better adapted to their environment. Crossover splits chromosomes and combines the genes from different chromosomes to form a new one. Mutation will

randomly change the data in the genes.

In nature environmental factors decide how an organism will survive. It can either adapt and pass on it's genetic information or it dies. GAs try to work in a similar manner by using a fitness function. The fitness function scores a chromosome's ability to provide a solution to a problem.

A selection operator is used to determine which chromosomes will pass on their genetic information based on their fitness scores. There are several different techniques to decide which chromosomes will contribute to the next generation. The technique that was used in this thesis was tournament selection. Tournament selection is a fast way of selecting chromosomes to provide genetic information. Another strong point of tournament selection is its ability to maintain diversity in the population, which can help in finding optimal solution. It works by randomly selecting two individuals from the population. It then chooses a value for $r$ between 0 and 1. Using a parameter $k$ set previously, it chooses the fitter of the two individuals if $r < k$. [9, pp. 170-171]

## A.2 GAlib

The genetic algorithm library chosen to implement the function optimizer was GAlib[10]. It is a powerful and free implementation of a C++ library to write solutions using genetic algorithms. It has several chromosome encoding techniques. The one used for this thesis was the real number encoding. This encoding provided an abstract interface that made it easy to use real numbers.

GAlib also provides interfaces where parameters are used to specify the rate of mutation and crossover, the number of generations and the population size. This made it very easy to setup a solution generator.

## A.3 GA Parameters

The values used to setup the genetic algorithm are given in table below.

| Parameter | Value |
|---|---|
| Generations | 100 |
| Population | 30 |
| Crossover | 0.8 |
| Mutation | 0.01 |

## A.4 Fitness Function

The following C++ code is the Objective scoring function that tells the GA how to score the chromosomes for fitness. The coefficients to be optimized are stored in the GAGenome data structure. The right hand side of the trigger function is evalutated in another function call.

Objective Function

```
//The Objective Function defines how the solution is scored.
float Objective(GAGenome& g)
{
  GABin2DecGenome & genome = (GABin2DecGenome &)g;
  vector<simEvent> *simptr =
               static_cast<vector<simEvent> *>(g.userData());
  vector<simEvent> &simVector = *simptr;
  int numberOfSims = simVector.size();
  float score = 0.0;
  float function = 0.0;
  float valueZ = genome.phenotype(3);
  int numberEvents = 0;
  int countNumCut = 0;

  for (int j = 0; j < numberOfSims; j++){
    numberEvents = simVector[j].get_numevents();
    //Counts the number cut for each group.
    countNumCut = 0;
    for (int i = 0; i < numberEvents; i++){
      function = evalFunction( g, simVector[j].get_event(i));
```

```
      //if the function <= valueZ it means to cut the event

      //Test to see it is a bad event and gets cut
      //Score +1
      if ( (function <= valueZ) &&
                (simVector[j].get_isGood() == 0) ){
        score += 1.0;
        countNumCut++;
      }
      //Test to see it is a good event and doesn't get cut
      //Score +5
      else if ( (function > valueZ) &&
                (simVector[j].get_isGood()==1)){
        score += 5.0;
      }
      //Test to see it is a good event and does get cut
      //Score -50
     else if ( (function <= valueZ) &&
                (simVector[j].get_isGood()==1)){
       countNumCut++;
       if (score < 50)
         score = 0;
       else
         score -= 50.0;
     }
    } //for (int i = 0; i < numberEvents; i++)

    //Check to see how many of the good
    //events got cut and reduce thescore.
    //If the number cut is above the threshold
    //then reset score to zero.  The goal
    //is to not cut good ones.
    if ((simVector[j].get_isGood()==1) &&
         (countNumCut > 50) ) {
      score = 0;
    }
  } //for (int j=0; j < numberOfSims; j++)
  return score;
}
```

# Appendix B

# Java Analysis Studio

Java Analysis Studio (JAS) is a relavtively new tool for performing data analysis and displaying results. Its focus group is nuclear and particle physicists, but it provides an easily extensible architecture that can be used to suppport analysis of data in any structured format. It has the ability to produce scatter plots, histograms, and lego plots.

## B.1 Data Interface Module

A Data Interface Module (DIM) is an extension to JAS that enables it to read new data formats. There are five files required by JAS in addition to the other class files that are needed store data required.

The *plugins.txt* is used by JAS to find the register code.

JAS-inf/plugins.txt

```
jasext.halldxml.Register
```

*Register.java* registers the DIM by passing the name of the local dim class. It also provides methods to allow JAS to display information about the DIM.

Register.java

```
package jasext.halldxml;

import jas.plugin.*;

public class Register extends ExtensionPlugin
                      implements IPluginInfo
{
   public void init()
   {
      registerDIM(new HallDXMLLocalDIM());
   }
   public String getName()
   {
      return "HallDXMLLocalDIM";
   }
   public String getVersion()
   {
      return "1.0.0";
   }
}
```

The class file *HallDXMLLocalDIM.java* creates a new event source object. It also provides information about file extensions.

HallDXMLLocalDIM.java

```
package jasext.halldxml;

import jas.jds.module.*;
import hep.analysis.EventSource;

/**
 * A LocalDIM for accessing Hall D XML files from a JAS client
 */
public class HallDXMLLocalDIM
        extends  BasicLocalDIM
{
/**
 * empty constructor
 */
   public HallDXMLLocalDIM()
   {
        super("Hall D XML File (*.xml)",".xml");
   }
```

```
/**
 * openDataSet opens the file specified by the dataSource
 * and returns it as an EventSource
 * @param dataSource the String name of the file
 * @returns EventSource the EventSource derived from the file
 */
   public EventSource openDataSet(String dataSource)
                           throws ModuleException
   {
        EventSource theEventSource =
                    new HallDXMLEventSource(dataSource);
        return theEventSource;
   }

/**
 * override toString to return name of this DIM
 */

   public String toString()
   {
        return("Hall D XML DIM");
   }

}// end class
```

*HallDAnalyzablePhysicsEvent.java* is used by the analysis classes that the user creates to access the event data.

HallDAnalyzablePhysicsEvent.java

```
package jasext.halldxml;

import java.util.*;

import hep.analysis.EventData;

public class HallDAnalyzablePhysicsEvent
    implements EventData
{
        PhysicsEvent theEvent = null;
        public HallDAnalyzablePhysicsEvent(PhysicsEvent ev)
        {
           theEvent = ev;
        }

        public PhysicsEvent getPhysicsEvent()
```

```
        {
           return theEvent;
        }

}// end class
```

The event source class loads the data into the objects. Due to the large size of the data files *HallDXMLEventSource.java* reads each event's data separately instead of loading all of the data at once.

HallDXMLEventSource.java

```
package jasext.halldxml;

import java.io.*;
import java.util.List;
import java.util.Iterator;

import jas.job.*;
import jas.jds.module.*;
import hep.analysis.*;

/**
   A class for opening a Hall D XML file as an event source.
   @author Dr. Dave Doughty James Hubbard
*/
public class HallDXMLEventSource implements EventSource
{
        protected String filename = null;
        protected int numberOfEvents = -1;
        protected PhysicsEvent nextEvent = null;
        protected HallDAnalyzablePhysicsEvent nextAnalyzableEvent
                                              = null;
        protected int currentEventNumber = 0;

    protected File xmlEvents = null;
    protected BufferedReader fIn = null;

        private int particleCount = 0;
        private String [] particleTypes = null;

        /**
         * Create a HallDXMLEventSource connected to the XML file
         * specified.
         * @param fileName the Hall D XML data file to open
```

```
 */
public HallDXMLEventSource(String filename)
                              throws ModuleException
{
        this.filename = filename;
        //System.out.println("filename is ="+filename);
        init();
}

/**
 * Open a Hall D XML Event Data File and prepare it for
 * analysis
 */
public void init()
{
        // Open the xml file
        try
        {
                xmlEvents = new File(filename);
                fIn = new BufferedReader
                             (new FileReader(xmlEvents));
                numberOfEvents = countEvents();
                fIn.close();
                xmlEvents = new File(filename);
                fIn = new BufferedReader
                              (new FileReader(xmlEvents));
}
        catch (Exception e)
        {
                System.out.println("Problem getting events
                               from file - error was ->"+e);
        }

        // Set the current event pointer
        currentEventNumber = 0;
}

/**
 * Return the string name of the file
 */
public String getName()
{
        int i = filename.lastIndexOf(
                    System.getProperty("file.separator"));
        return filename.substring(i+1);
}
```

```
/**
 * Return the class of the objects returned by
 * getNextEvent
 * @return Class the class of the
 * HallDAnalyzablePhysicsEvent
 */
public Class getEventDataClass()
{
   return HallDAnalyzablePhysicsEvent.class;
}

/**
 * return the number of events
 * @return int i the number of events
 */
public int getTotalNumberOfEvents()
{
        return numberOfEvents;
}

/**
 * return the next event
 * @return EventData the next event
 */
public EventData getNextEvent() throws EndOfDataException
{
        PhysicsEvent nextEvent =
                 getNextPhysicsEventFromFile();
        currentEventNumber += 1;
        if(nextEvent != null)
        {
           nextAnalyzableEvent =
             new HallDAnalyzablePhysicsEvent(nextEvent);
              return nextAnalyzableEvent;
        }
        else
        {
                throw new EndOfDataException();
        }
}

/**
 * Returns a physics event from a file.  The events are read
 * only when called.  Reading all events into memory at once
 * will cause an out of memory error.
 * @return return physicsEvent
 */
```

```
private PhysicsEvent getNextPhysicsEventFromFile ()
{
        String line = null;
        String beginTag = "<physicsEvent";
        String endTag = "</physicsEvent>";
        boolean inPhysicsEvent = false;
        PhysicsEvent event = null;
        ByteArrayOutputStream baos =
                    new ByteArrayOutputStream();
        PrintWriter out = new PrintWriter (baos, true);
        ByteArrayInputStream  bais = null;
        int linecount = 0;
        boolean getEvent = false;

        try
        {
           //While loop reads data until eof or
           // it's gotten an event.
           //getEvent needs to go first, relies on
           //shortcircuiting test to
           //prevent reading of next line.
           while ( !getEvent &&
                     (line=fIn.readLine()) != null )
           {
              linecount += 1;
              if ( (!inPhysicsEvent) &&
                   (line.indexOf(beginTag) >= 0)  )
              {
                     inPhysicsEvent = true;
                   out.println (line);
              }
              else if (inPhysicsEvent)
              {
                 out.println(line);
                 if (line.indexOf(endTag) >=0)
                 {
                    inPhysicsEvent = false;
                    getEvent = true;
                 }
               }
             } //while ( ( line=fIn.ReadLine()) != null)
        }
        catch (Exception eFinReadLine)
        {
           System.out.println(
            "Exception reading line getNextPhyscisEvent: " );
           System.out.println(eFinReadLine);
```

```
            System.out.println("Linecount: " + linecount);
        }

        //Test to see if an event was read.
        //If it was read then unmarshal it.
        if ( baos.size() > 0 )
        {
            byte[] bArray = baos.toByteArray();
            bais = new ByteArrayInputStream(bArray);
            try
            {
                 event = event.unmarshal(bais);
                 baos.close();
                 bais.close();
            }
            catch (Exception eUnmarshal)
            {
               System.out.println ("Exception Unmarshalling
                       data in getNextPhysicsEvent:  ");
               System.out.println (eUnmarshal);
               System.out.println ("Linecount: " + linecount);
            }
        }
        return event;
}

//This method counts the number of events from the file fIn.
//Relies on scanning the file.
private int countEvents()
{
        int counter = 0;
        String line;
        int numberOfProducts = 0;
        int counterParticles = 0;

        try
        {
           while ( (line=fIn.readLine()) != null )
           {
              if ( counter == 0 ) {
                 if(line.indexOf("<product") >= 0)
                   numberOfProducts++;
              }

              if ( (counter == 1) &&
                    (particleTypes == null) ) {
                 particleTypes = new String[numberOfProducts];
```

```
                }

                if ( (counter == 1) &&
                     (line.indexOf("<product") >= 0)) {
                   particleTypes[counterParticles] =
                       findParticleType(line);
                   counterParticles++;
                }

                if (line.indexOf("</physicsEvent>") >=0 ){
                   counter++;
                }
             }
          }
          catch (Exception eFinReadLine)
          {
             System.out.println("Exception reading
                     line in countEvents " + eFinReadLine);
          }
          particleCount = numberOfProducts;
          return counter;
} // private int countEvents()

/**
 * Returns a string particle type from a line of input.
 * @param line line read from a file or other input
 * @return a particle type as a string
 */

private String findParticleType(String line)
{
          int begin = 15;
          int end = 0;
          String type = null;

          begin += line.indexOf("<product");
          end = line.indexOf("\"", begin);
          type = line.substring(begin, end);
          //System.out.println ("Particle Type: " + type);

          return type;
}

public String [] getParticleTypes ()
{
          return particleTypes;
}
```

```
        public int getParticleCount()
        {
                return particleCount;
        }


        /**
         * This is called before analyzing the first event
         */
        public void beforeFirstEvent()
        {
        }

        /**
         * This is called after analyzing the last event
         */
        public void afterLastEvent()
        {
        }

        /**         * Close the EventSource and free all resource
         */
        public void close()
        {
        }

        public void finalize()
        {
                 close();
        }

}// end class
```

## B.2 Analysis Code

The following code section is an analysis file that displays historgrams of energy and track counts for various detector components.

N3PiAnalysis.java

```
import hep.analysis.*;
import hep.analysis.partition.*;
```

```
import jasext.halldxml.*;
import java.util.*;
import java.io.*;

final public class N3PiAnalysis extends EventAnalyzer
{

    final private float PI = (float)Math.PI;

    private int NUMBER_PARTICLES = 3;
    private String [] particleTypes = {"unk", "unk", "unk"};

    private float ENERGY_MIN=(float) -0.3;
    private float ENERGY_MAX=(float) 9.1;
    private int ENERGY_PART = (int)((ENERGY_MAX - ENERGY_MIN)*10);

    final private FixedPartition2D test2DPart =
        new FixedPartition2D(0, PI, 100, 0, PI, 100);
    final private FixedPartition thetaStd =
        new FixedPartition (0, PI, 100);

    private Histogram thetaMin = null;
    private Histogram thetaMiddle = null;
    private Histogram thetaMax =null;
    private Histogram minMax = null;

    private Histogram [] pAngle = null;
    private FixedPartition2D partEnergyVsEnergy= null;
    private FixedPartition  partEnergy = null;
    private Histogram forwardEMcalEnergy = null;
    private Histogram barrelEMcalEnergy = null;
    private Histogram forwardEnergyBarrelEnergy = null;
    private Histogram totalCalEnergy = null;
    private Histogram numberOfTrackstofPoint = null;
    private Histogram numberOfTracksStartPoint = null;
    private Histogram totalNumTrackstofPointStartPoint = null;
    private Histogram tracksTofPointStartPoint = null;
    private Histogram histEnergyPerTrack = null;
    private Histogram trackBarEnergyStart = null;
    private Histogram trackForEnergyStart = null;
    private FixedPartition2D particleCalEnergyPart = null;

    private Histogram [] energyParticleForCal = null;
    private Histogram [] energyParticleBarCal = null;

    final private FixedPartition paritionEnergyDepositions =
        new FixedPartition (-0.3, 1.1, 140);
```

```
private Histogram [] energyDepositions = null;

final private Histogram histUniqueParticles =
    new Histogram ("Good (1) & Bad(-1) Event Count");

private BarrelAnalysis theBarrel = null;
private ForwardCalAnalysis theForwardCal = null;
private ParticleAnalysis particle = null;
private ForwardTOFAnalysis theForwardTOF = null;
private StartCntrAnalysis theStartCntr = null;


// To group these histograms into folders,
// declare some variables of type HistogramFolder,
// initialize those folders in the consructor, and
// pass the appropriate folder to the Histogram and
// ScatterPlot constructors.

public N3PiAnalysis()
{
    // Enter constructor code here.

    NUMBER_PARTICLES = 3;

    particleTypes = new String [] {"pi+", "pi-", "pi+"};
    ENERGY_MIN=(float) -0.3;
    ENERGY_MAX=(float) 9.1;
    ENERGY_PART = (int)((ENERGY_MAX - ENERGY_MIN)*10);

    theBarrel= new BarrelAnalysis(NUMBER_PARTICLES);
    theForwardCal = new ForwardCalAnalysis(NUMBER_PARTICLES);
    particle = new ParticleAnalysis (NUMBER_PARTICLES);
    theForwardTOF = new ForwardTOFAnalysis(NUMBER_PARTICLES);
    theStartCntr = new StartCntrAnalysis (NUMBER_PARTICLES);

    thetaMin =
        new Histogram("Angle: thetaMin",
                      thetaStd.makeCopy());
    thetaMiddle =
        new Histogram("Angle: thetaMiddle",
                      thetaStd.makeCopy());
    thetaMax =
        new Histogram("Angle: thetaMax",
                      thetaStd.makeCopy());
    minMax =
        new Histogram2D ("Angle: thetaMin(y) vs thetaMid(x)",
```

```
                        test2DPart);

particleCalEnergyPart =
    new FixedPartition2D (ENERGY_MIN, ENERGY_MAX,
                          ENERGY_PART, ENERGY_MIN,
                          ENERGY_MAX, ENERGY_PART);

partEnergyVsEnergy =
    new FixedPartition2D
    (ENERGY_MIN, ENERGY_MAX, ENERGY_PART,
     ENERGY_MIN, ENERGY_MAX, ENERGY_PART);

partEnergy = new FixedPartition (ENERGY_MIN,
                                 ENERGY_MAX, ENERGY_PART);

pAngle = new Histogram[NUMBER_PARTICLES];
for (int i = 0; i < NUMBER_PARTICLES; i++){
    pAngle [i]=
        new Histogram("Angle: Particle" + (i+1)
                      +" ("+ particleTypes[i]+ ") ",
                      thetaStd.makeCopy());
}

forwardEMcalEnergy =
    new Histogram("Energy Cal: Forward EM Cal E",
                  partEnergy.makeCopy());
barrelEMcalEnergy =
    new Histogram("Energy Cal: Barrel EM Cal E",
                  partEnergy.makeCopy());

forwardEnergyBarrelEnergy =
    new Histogram2D ("Energy Cal: Forward Cal(y)
                             vs Barrel Cal(x)",
                     partEnergyVsEnergy.makeCopy());
totalCalEnergy =
    new Histogram ("Energy Cal: Total in Both
                               Calorimeters",
                   partEnergy.makeCopy());

numberOfTrackstofPoint =
    new Histogram ("Tracks: ForwardTOF");
numberOfTracksStartPoint =
    new Histogram ("Tracks: StartCntr");
totalNumTrackstofPointStartPoint =
    new Histogram ("Tracks: Total forwardTOF and StartCntr");
tracksTofPointStartPoint =
    new Histogram2D ("Tracks: ForwardTOF(y) vs StartCntr(x)",
```

```
                    new FixedPartition2D
                    (0, 4.1, 50, 0, 4.1, 50) );
histEnergyPerTrack =
    new Histogram2D("Tracks: Energy vs Track Count",
                   new FixedPartition2D
                   (0, 6.1, 100,
                    ENERGY_MIN, ENERGY_MAX, 100));
trackBarEnergyStart =
    new Histogram2D("Tracks: Barrel Cal
                     Energy vs Track Count",
        new FixedPartition2D(0, 3.1, 100,
                             ENERGY_MIN,
                             ENERGY_MAX,
                             100));
trackForEnergyStart =
    new Histogram2D(
                   "Tracks: Forward Cal
                    Energy vs Track Count",
        new FixedPartition2D(0, 3.1, 100,
                            ENERGY_MIN,
                            ENERGY_MAX,
                            100));
energyDepositions = new Histogram [NUMBER_PARTICLES];

for (int i =0; i < NUMBER_PARTICLES; i++){
    energyDepositions[i] =
        new Histogram("Energy Deposition: Forward Cal Particle"
                      + (i+1)+ " - " + particleTypes[i],
                      paritionEnergyDepositions.makeCopy());
}

energyParticleForCal = new Histogram2D [NUMBER_PARTICLES];
energyParticleBarCal = new Histogram2D [NUMBER_PARTICLES];

for (int i=0; i < NUMBER_PARTICLES; i++){
    energyParticleForCal[i] =
        new Histogram2D("Energy Forward Cal: Initial Particle"
                        + (i+1) +" ("
                        + particleTypes[i]+
                        ")   vs Forward EM Cal(x)",
                        partEnergyVsEnergy.makeCopy() );
    energyParticleBarCal[i] =
        new Histogram2D("Energy Barrel Cal: Initial Particle"
                        + (i+1) +" (" + particleTypes[i]+
                        ") vs Barrel Cal(x)",
                        partEnergyVsEnergy.makeCopy() );
}
```

```
}// public Analysis() constructor


public void beforeFirstEvent()
{
}

public void afterLastEvent()
{
}

public void processEvent(final EventData d)
{
     final jasext.halldxml.HallDAnalyzablePhysicsEvent data =
        (jasext.halldxml.HallDAnalyzablePhysicsEvent) d;

    PhysicsEvent nextEvent = data.getPhysicsEvent();
    //histAngles(nextEvent);
    //Calculation for Histogram here

    theBarrel.reloadBarrel(nextEvent);
    theForwardCal. reloadForwardCal(nextEvent);
    particle.reloadParticle(nextEvent);
    theForwardTOF.reloadForwardTOF(nextEvent);
    theStartCntr.reloadStartCntr(nextEvent);

    float forwardCalEnergy =
        theForwardCal.getEnergy();
    float forwardCalCheatEnergy =
        theForwardCal.getCheatEnergy();
    float barrelCalEnergy =
        theBarrel.getEnergy();
    float barrelCalCheatEnergy =
        theBarrel.getCheatEnergy();
    float totalEnergyCalorimeters = 0;
    float [] barrelCalParticleEnergy =
        theBarrel.getParticleEnergy();
    float [] forwardCalParticleEnergy =
        theForwardCal.getParticleEnergy();


    float [] initalParticleEnergy =
        particle.getEnergy ();
    float [] minMaxAngles =
        particle.getThetaOrdered ();
    float [] thetaAngles =
        particle.getThetaAngles();
```

```
for (int i=0; i < NUMBER_PARTICLES; i++){
    pAngle[i].fill(thetaAngles[i]);
}
thetaMin.fill(minMaxAngles[0] );
thetaMiddle.fill(minMaxAngles[1] );
thetaMax.fill(minMaxAngles[2] );
minMax.fill(minMaxAngles[1],minMaxAngles[0]);

for (int i = 0; i < NUMBER_PARTICLES; i++) {
    if (forwardCalParticleEnergy[i]>= 0) {
        energyDepositions[i].fill(
              forwardCalParticleEnergy[i]
              /initalParticleEnergy[i]);
    }
    else {
        energyDepositions[i].fill(
              forwardCalParticleEnergy[i]/5);
    }
}

for (int i = 0; i < NUMBER_PARTICLES; i++){
    if (forwardCalParticleEnergy[i] == -1 ){
        energyParticleForCal[i].fill(
              forwardCalParticleEnergy[i]/5,
              initalParticleEnergy[i]);
    }
    else{
        energyParticleForCal[i].fill(
              forwardCalParticleEnergy[i],
              initalParticleEnergy[i]);
    }
    if (barrelCalParticleEnergy[i] == -1 ){
        energyParticleBarCal[i].fill(
              barrelCalParticleEnergy[i]/5,
              initalParticleEnergy[i]);
    }
    else{
        energyParticleBarCal[i].fill(
              barrelCalParticleEnergy[i],
              initalParticleEnergy[i]);
    }
}


totalEnergyCalorimeters = barrelCalCheatEnergy +
    forwardCalCheatEnergy;
```

```
        forwardEMcalEnergy.fill(forwardCalCheatEnergy);

        barrelEMcalEnergy.fill(barrelCalCheatEnergy);
        forwardEnergyBarrelEnergy.fill(barrelCalCheatEnergy,
                                       forwardCalCheatEnergy);

        totalCalEnergy.fill(totalEnergyCalorimeters);


        int numTracksInForwardTOF = theForwardTOF.getTrackCount();
        int numTracksInStartCntr = theStartCntr.getTrackCount();
        int totalNumberOfTracks = numTracksInForwardTOF +
            numTracksInStartCntr;

        trackBarEnergyStart.fill(numTracksInStartCntr,
                                 barrelCalCheatEnergy);
        trackForEnergyStart.fill(numTracksInForwardTOF,
                                 forwardCalCheatEnergy );
        numberOfTrackstofPoint.fill(numTracksInForwardTOF);
        numberOfTracksStartPoint.fill(numTracksInStartCntr);
        totalNumTrackstofPointStartPoint.fill( totalNumberOfTracks);
        histEnergyPerTrack.fill(totalNumberOfTracks,
                                totalEnergyCalorimeters);

        tracksTofPointStartPoint.fill(numTracksInStartCntr,
                                      numTracksInForwardTOF);

    } //public void processEvent

}// N3PiAnalysis
```

---

# REFERENCES


[1] Hall D Collaboration Board. *The Science of Quark Confinement and Gluonic Excitations: Gluex/Hall D Design Report.* Technical report, Jefferson Laboratory, Newport News, VA. Nov. 2002.

[2] P. Eugenio. *Genr8*: A general monte carlo event generator. Technical report, Carnegie Mellon University, 1998.

[3] R. Jones, *HDDM - HALL D Data Model*, Web page, http://zeus.phys.uconn.edu/halld/datamodel/doc. University of Connecticut, May, 2001.

[4] R. Jones, 2001. The HDGeant Monte Carlo Program. University of Connecticut

[5] Sun Microsystems, Inc., *Java Architecture for XML Binding*, Web page, http://java.sun.com/xml/jaxb/.

[6] CERN, *Physics Analysis Workstattion*, Computer Software, http://wwwinfo.cern.ch/asd/paw/.

[7] T. Johnson et al, *Java Analysis Studio*, Computer Software, http://jas.freehep.org.

[8] E. Wolin, Private Communication.

[9] M. Mitchell, *An Introduction to Genetic Algorithms*, (The MIT Press, Cambridge, 1998).

[10] M. Wall, *GAlib: Genetic Algorithm Library*, Computer software, http://lancet.mit.edu/ga/.