

In presenting this thesis in partial fulfillment of the requirements for an advanced degree at Idaho State University, I agree that the Library shall make it freely available for inspection. I further state that permission for extensive copying of my thesis for scholarly purposes may be granted by the Dean of the Graduate School, Dean of my academic division, or by the University Librarian. It is understood that any copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Signature \_\_\_\_\_

Date \_\_\_\_\_

Rastered Beam Interaction Vertex Reconstruction in the CLAS12 Detector

by

David Friant

A thesis

submitted in partial fulfillment

of the requirements for the degree of

Master of Science in Physics

Idaho State University

Summer 2019

To the Graduate Faculty:

The members of the committee appointed to examine the thesis of David Friant find it satisfactory and recommend that it be accepted.

---

Dr. Tony Forest,  
Major Advisor

---

Dr. Dan Dale,  
Committee Member

---

Dr. David Beard,  
Graduate Faculty Representative

# Contents

<b>Abstract</b> . . . . .	<b>vii</b>
<b>1 Introduction</b> . . . . .	<b>1</b>
<b>2 Theory</b> . . . . .	<b>4</b>
2.1 Drift Chambers . . . . .	4
2.1.1 Typical Construction and Principle of Operation . . . . .	4
2.1.2 Addition of a Magnetic Field . . . . .	6
2.1.3 CLAS12 . . . . .	6
2.2 Track Reconstruction Elements . . . . .	10
2.2.1 Coordinate Systems . . . . .	10
2.2.2 Runge-Kutta Approximation . . . . .	12
2.2.3 Kalman Fitter . . . . .	13
2.3 Rastered Beam . . . . .	16
<b>3 Simulation and Reconstruction</b> . . . . .	<b>17</b>
3.1 Physical Setup . . . . .	17
3.2 Simulation . . . . .	18
3.3 Current Reconstruction Algorithms . . . . .	20
3.4 Proposed Reconstruction Algorithm . . . . .	21
3.4.1 Parameters . . . . .	22
3.4.2 Overarching Algorithm . . . . .	23
3.4.3 Reconstruction . . . . .	23

3.4.4	Grid Search . . . . .	23
3.4.5	Vertex Search . . . . .	24
<b>4</b>	<b>Analysis . . . . .</b>	<b>25</b>
4.1	$\Delta Z$ Comparisons . . . . .	26
4.2	$\Delta Z$ Summary Table . . . . .	31
4.3	$\Delta Z$ Dependence on Raster Radius . . . . .	32
4.4	$\Delta X$ and $\Delta Y$ . . . . .	33
4.5	Efficiency . . . . .	36
<b>5</b>	<b>Conclusion . . . . .</b>	<b>37</b>
	<b>References . . . . .</b>	<b>39</b>
	<b>Appendix A Example Program Calls and Example GCard . . . . .</b>	<b>42</b>
A.1	Example GCard . . . . .	42
A.2	Program Calls . . . . .	47
	<b>Appendix B Descriptions of Hit Based and Time Based Reconstructions . . . . .</b>	<b>48</b>
B.1	Definition of Terms . . . . .	48
B.2	Hit Based Algorithm . . . . .	49
B.2.1	Hits to Clusters . . . . .	49
B.2.2	Clusters to Segments . . . . .	49
B.2.3	Segments to Crosses . . . . .	50
B.2.4	Crosses to Tracks Part I . . . . .	50
B.2.5	Crosses to Tracks Part II . . . . .	52
B.3	Time Based Algorithm . . . . .	52
B.3.1	Hits to Clusters . . . . .	52
B.3.2	Clusters to Segments . . . . .	53
B.3.3	Reconstructing Tracks . . . . .	53

Appendix C Source Code For Raster Based Reconstruction . . . . .	54
--	----

## Abstract

### Rastered Beam Interaction Vertex Reconstruction in the CLAS12 Detector

The thesis reports the results of an improved track reconstruction program that is used to determine the momentum and interaction vertex of particles scattering in experiments performed using the CLAS12 Detector at Jefferson Lab. Particles are reconstructed from the CLAS12 drift chamber and scintillator detector sub-systems. The improved algorithm performs a three dimensional grid search in momentum space in an attempt to minimize the distance of closest approach between the particle's path and the transverse location of the incident electrons that scatter from nucleons in a cryogenic target. Analysis has shown that the new algorithm is capable of improving the reconstruction in the X and Y directions (orthogonal to the incident electrons) by a factor of no less than ten, while at least maintaining the quality of the reconstruction in the Z direction for a subset of input events.

Key Words: CLAS12, Rastered Beam, Reconstruction

# Chapter 1

## Introduction

The CLAS12 is a charged and neutral particle detector constructed in Jefferson Lab's Hall B. The detector is used to reconstruct nuclear physics interactions that result when GeV energy electrons interact with select targets. CLAS12's precursor, the original CLAS (CEBAF Large Acceptance Spectrometer), was built to facilitate measurements of excited states of nucleons and characterize nucleon-nucleon correlations. Contemporaneous with the 12GeV upgrade to Jefferson Lab's accelerator, CLAS was removed and CLAS12 was built in its place.<sup>1</sup>

CLAS12 is designed to not only continue the mission of its predecessor, but to also measure generalized parton distributions of nucleons and search for exotic forms of matter. It is currently slated to perform experiments for more than ten years into the future and will likely continue to contribute even after that.<sup>1</sup> Figure 1.1 shows a model of the detector with labels detailing some of the sub detector's and other components. It is important to note here that CLAS12 is not a singular detector, but instead a suite of detectors which can be correlated together in order to more precisely reconstruct the interactions that occur within the physics target.

Alongside the construction of CLAS12, several software packages have been created to handle simulating the detector response, reconstructing events from detector the detector response, and visualizing the reconstructed interactions. For simulation, the CLAS Collab-



oration has developed a package that provides all of the necessary information to GEMC<sup>2</sup> (GEant4 Monte-Carlo), a general program designed to make interfacing with Geant4<sup>3</sup> easier. The program used for viewing events is called CED (CLAS Event Display), and is a continuation of a program originally designed for use with the CLAS. Most importantly for this document, as it is the topic of discussion, is the reconstruction software. Known as clas12-offline-software<sup>4</sup>, it is a collection of various (mostly) java programs used to process the data collected from CLAS12 and reconstruct the observed particle mass, momentum, and location within the detector.

This thesis is two-fold in purpose. First, it is intended to provide an understanding of the current methods used to reconstruct particle trajectories from the drift chamber sub-detectors of CLAS12. This includes both a theoretical understanding of the underlying algorithms and physics as well as the actual implementation of those ideas. The second purpose is to present an additional method of reconstruction which takes the already existing drift chamber reconstruction output and further refines it so that interactions which arise from a rastered beam may be reconstructed more accurately.

It should be noted that, in the context of this thesis, the primary motivation behind the development of this new reconstruction algorithm is to allow for the reconstruction of events from a physics target that contains two polarized target regions separated by some distance. More details will be given about the target in the Simulation and Reconstruction chapter. Semi-Inclusive Deep Inelastic Scattering (SIDIS) events are of particle interest. SIDIS events only consider the scattered incident particle, an electron in this case, and the highest energy hadron produced by the interaction, if any. The core idea is that when the scattered particle interacts with a parton in the struck particle, there will be a primary product hadron accompanied by a hadron shower. The kinematics of this primary product and the scattered particle are believed to encode most of the information about the struck parton.

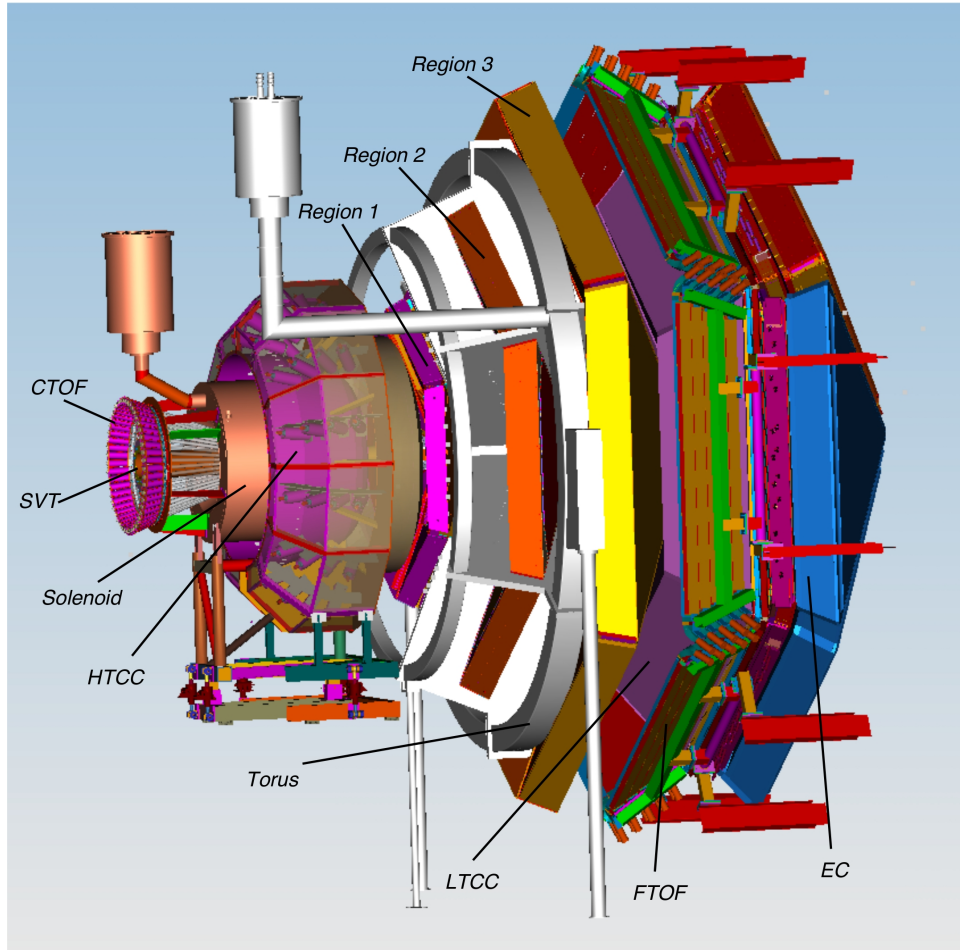


Figure 1.1: A 3-D model of the entire CLAS12 detector. The components marked as Regions 1, 2, and 3 indicate the drift chambers.<sup>5</sup>

# Chapter 2

## Theory

This chapter is meant to provide sufficient knowledge and background to understand the general workings of the CLAS12 drift chambers, the most critical elements of the reconstruction process, and the rasterization of the scattered electrons. With regards to the drift chambers, particular attention is paid to the geometry and physical construction as that directly effects the acceptance of the detector. The elements of the most importance discussed here are the concepts of Runge-Kutta approximations and Kalman Fitters. Both are used heavily in the reconstruction process. And finally, the discussion about rastered beams gives background as to why this project exists to begin with.

### 2.1 Drift Chambers

The CLAS12 drift chambers are designed to determine the momentum of charged particles moving through them. The following sections provide a brief overview of drift chambers in general.

#### 2.1.1 Typical Construction and Principle of Operation

Drift chambers are charged particle detectors that are designed to measure the trajectory of a particle's path through the detector. The trajectory is backed out from discrete position

and momentum data from wires within the chamber. This process is typically called Inverse Kinematics.

Typically, drift chambers are constructed out of a chamber filled with an ionizable gas and planes of conducting wire. The wire planes are arranged to be ideally perpendicular to the particle's direction of motion and come in two alternating types: cathode planes and anode (sense) planes. The planes are designed to create a nearly uniform static electric field which will drive electrons towards the sense planes. As described in the next paragraph, the sense wires collect ionized electrons, producing an electronic signal. That electronic signal propagates in both directions along the sense wire and away from the location of the ionization. The signal's propagation is measured by a pair of coupled timers at either end of the wire. It is common to have multiple sets of layers which are oriented differently, though still perpendicularly to the direction of motion, to increase the accuracy of the reconstructed trajectories.<sup>6</sup>

The general principle of producing an electronic signal in a drift chamber is as follows. A charged particle moves through the drift chamber and ionizes atoms along its path. The electrons, which have been separated from these atoms, are now accelerated towards the anode wires (sense wires). If the anode wire is very thin, the electric field near to it becomes very strong, causing the electrons to pick up a great deal of speed and cause a Townsend avalanche. A Townsend avalanche is a cascading series of ionizations which help to amplify the signal.<sup>7</sup> This signal being a voltage pulse traveling towards either end of the wire from the point where the electrons hit. Using the coupled timers at the ends, it is then possible to use the difference in time of signal propagation to calculate the position along the wire of the hit.

This position along the axis of the wire is only one dimensional information about a particle traveling through 3D space. It is, however, possible to couple the timing information from multiple nearby sense wires and a measurement of the time the liberated electrons take to reach the sense wire to calculate the distance of closest approach (DOCA) to each of them. This then gives a position along the axis of each wire as well as a radius perpendicular to that

axis at that point. If all of this information is known perfectly, then the vector component of the path which lies in the plane of a circle defined by the aforementioned radius will be tangent to that circle. Combining that information with the change in hit position along the axis of each wire allows for the ultimate measurement of the particle's path through the detector.

### 2.1.2 Addition of a Magnetic Field

The inclusion of a magnetic field into a drift chamber allows for the reconstruction of not just the path of the particle, but also the magnitude of its momentum. A uniform magnetic field perpendicular to the particle's direction of motion, for example, would cause the path to bend into some section of a circle, thus changing the expected hit position along the wires of the sense planes. Using these hits, it is then possible to reconstruct the radius of curvature of the path. With the assumption the particle carries the usual elementary charge, it is then possible to calculate the particle's momentum by equating the central force and magnetic force as shown in equation 2.1.

$$\frac{\gamma m v^2}{r} = q v B \implies \gamma m v = q B r \quad (2.1)$$

### 2.1.3 CLAS12

CLAS12's drift chambers are the primary source of path information for interaction reconstruction within the entire detector suite and are the focus of this document. As such, it is important to make as clear as possible their construction.

#### CLAS12's Drift Chamber Construction

The CLAS12 Detector's drift chamber is broken up into 18 separate sub-chambers as shown in figure 2.1. There are six chambers in each of three regions which are approximately 2.1, 3.3, and 4.5 meters away from the physics target position. Within each region, the six

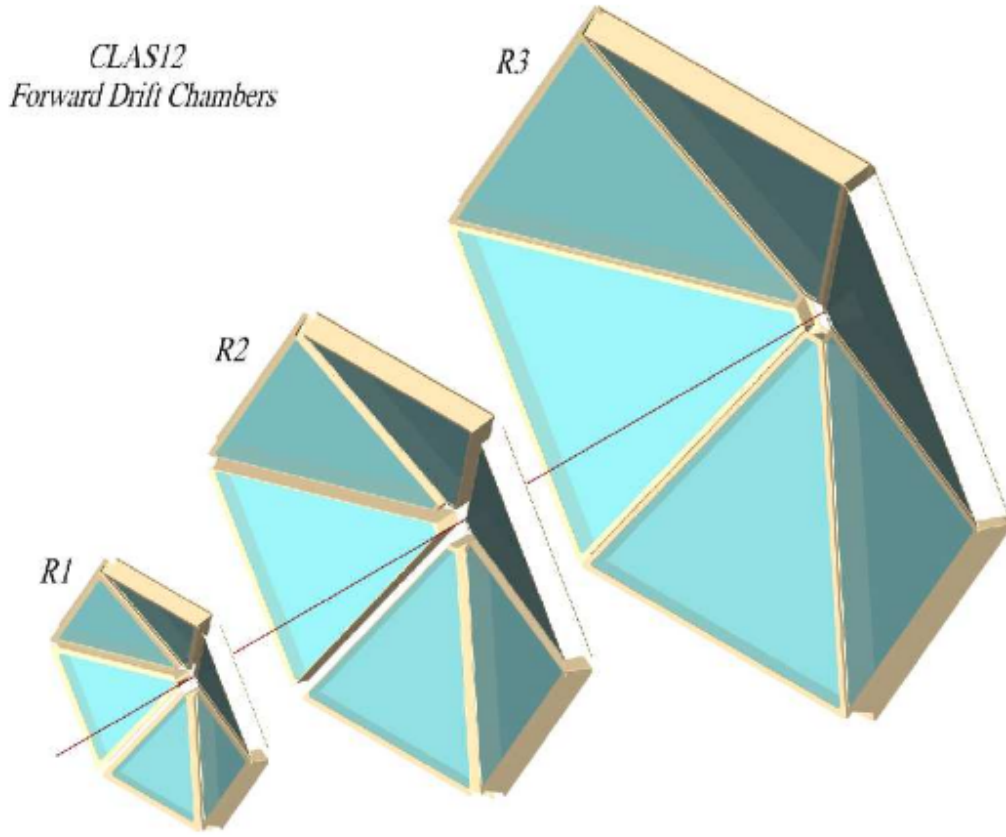


Figure 2.1: The three regions and six sectors per region in the drift chambers.<sup>5</sup>

triangular chambers are arranged to be in symmetric sectors around the beam axis with angular coverage between  $5^\circ$  and  $40^\circ$  as measured at the target location with respect to the beam axis. The design of the detector allows for a momentum resolution of less than one percent.<sup>8</sup>

Within each region and sector are two superlayers of wire planes which are arranged such that the axes of the wires lie in parallel planes that are rotated by  $12^\circ$  with respect to each other. Within each superlayer are 6 layers of sense wires. Each layer has 112 sense wires which have a hexagonal arrangement of cathode wires around them. These hexagonal arrangements are referred to as cells. They grow in size from region 1 to region 3 and provide a spatial resolution of approximately  $300\mu\text{m}$ .<sup>8</sup>

## Magnetic Fields

The CLAS12 detector has two magnetic fields, one solenoidal field at the target and one toroidal field centered around the second drift chamber region. The solenoidal field points primarily in the direction of the beamline with a central value of 5 Tesla, and is designed to prevent Möller electrons, a primary background, from entering the detector by redirecting their trajectories to be down the beamline.<sup>9</sup> The toroidal field is actually the result of six smaller coils, one in each sector, which all contribute to create a field that is primarily in the phi (azimuthal about the beam line) direction.<sup>10</sup> This is designed to bend particles either into or away from the beam line through the drift chambers which allows for the reconstruction of the particle's momentum. See figure 2.2.

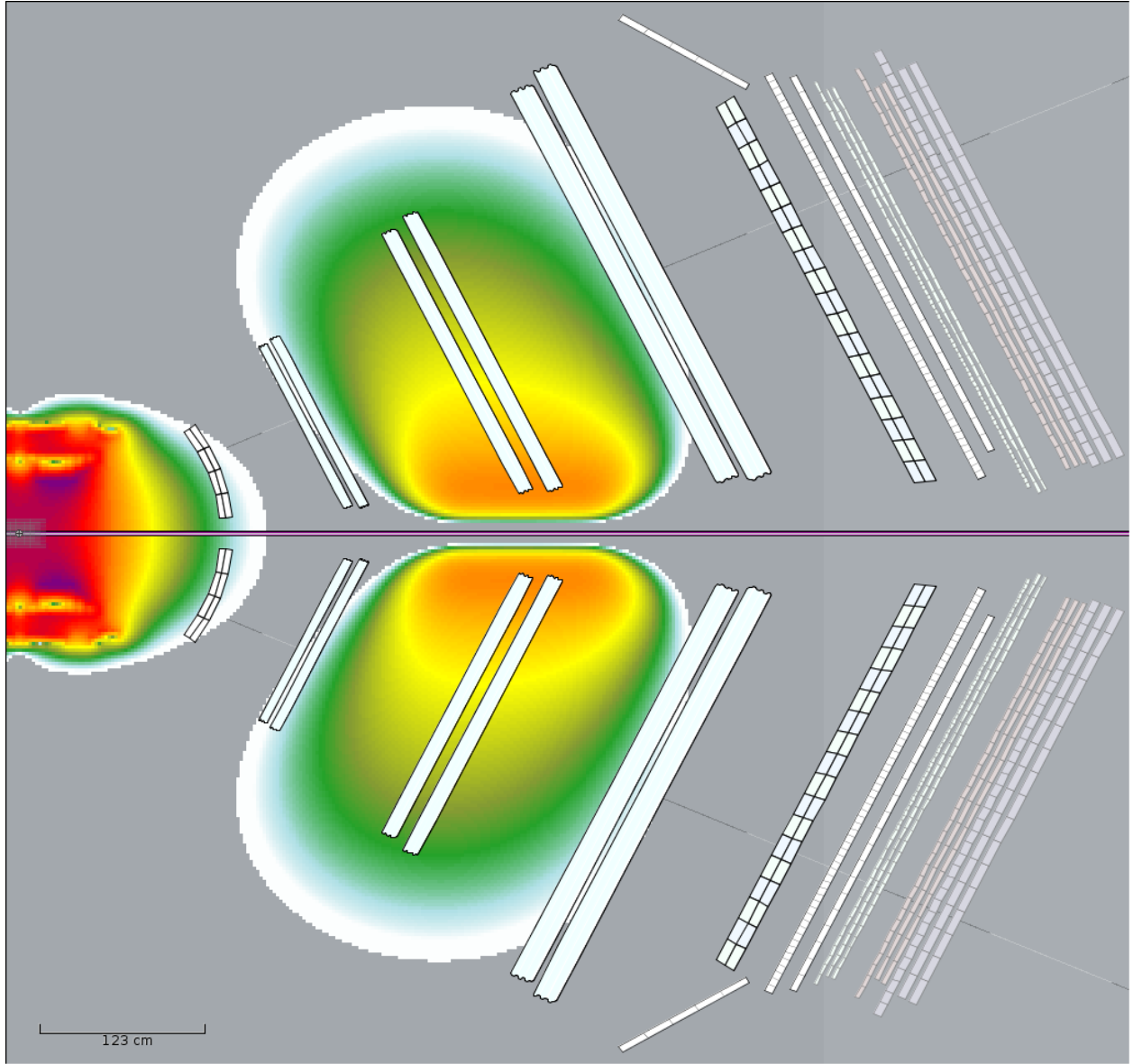


Figure 2.2: The strength of both the solenoidal (left) and toroidal (center) magnetic fields.



## 2.2 Track Reconstruction Elements

This section describes the theory behind and implementation of several of the more opaque components of the reconstruction algorithms.

### 2.2.1 Coordinate Systems

The collaboration has defined two coordinate systems in addition to the lab system to reduce the complexity of the reconstruction process within the drift chambers. The lab coordinate system is defined by a right-handed system such that the positive y-direction is upwards, against the pull of gravity and the positive z-direction is down the beam line. This results in the positive x-direction bisecting what is known as sector one as shown in figure 2.3.

The sector coordinate system is defined by rotating the sector of interest about the Z axis into the position of sector one as per figure 2.3. This rotation will naturally be some multiple of  $60^\circ$  and will only take into account  $\pm 30^\circ$  about the  $0^\circ$  angle. The tilted sector coordinate system takes a sector coordinate system and rotates it about the y-axis, again as per figure 2.3, by  $25^\circ$  such that the Z-direction is now perpendicular to the sector drift chambers. Transformations between these different coordinate systems are typically handled by rotation matrices. The matrices 2.2, 2.3, and 2.4 operate on vectors to rotate about the X, Y, and Z axes respectively.<sup>11</sup>

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta) & -\sin(\theta) \\ 0 & \sin(\theta) & \cos(\theta) \end{bmatrix} \quad (2.2)$$

$$\begin{bmatrix} \cos(\theta) & 0 & -\sin(\theta) \\ 0 & 1 & 0 \\ \sin(\theta) & 0 & \cos(\theta) \end{bmatrix} \quad (2.3)$$

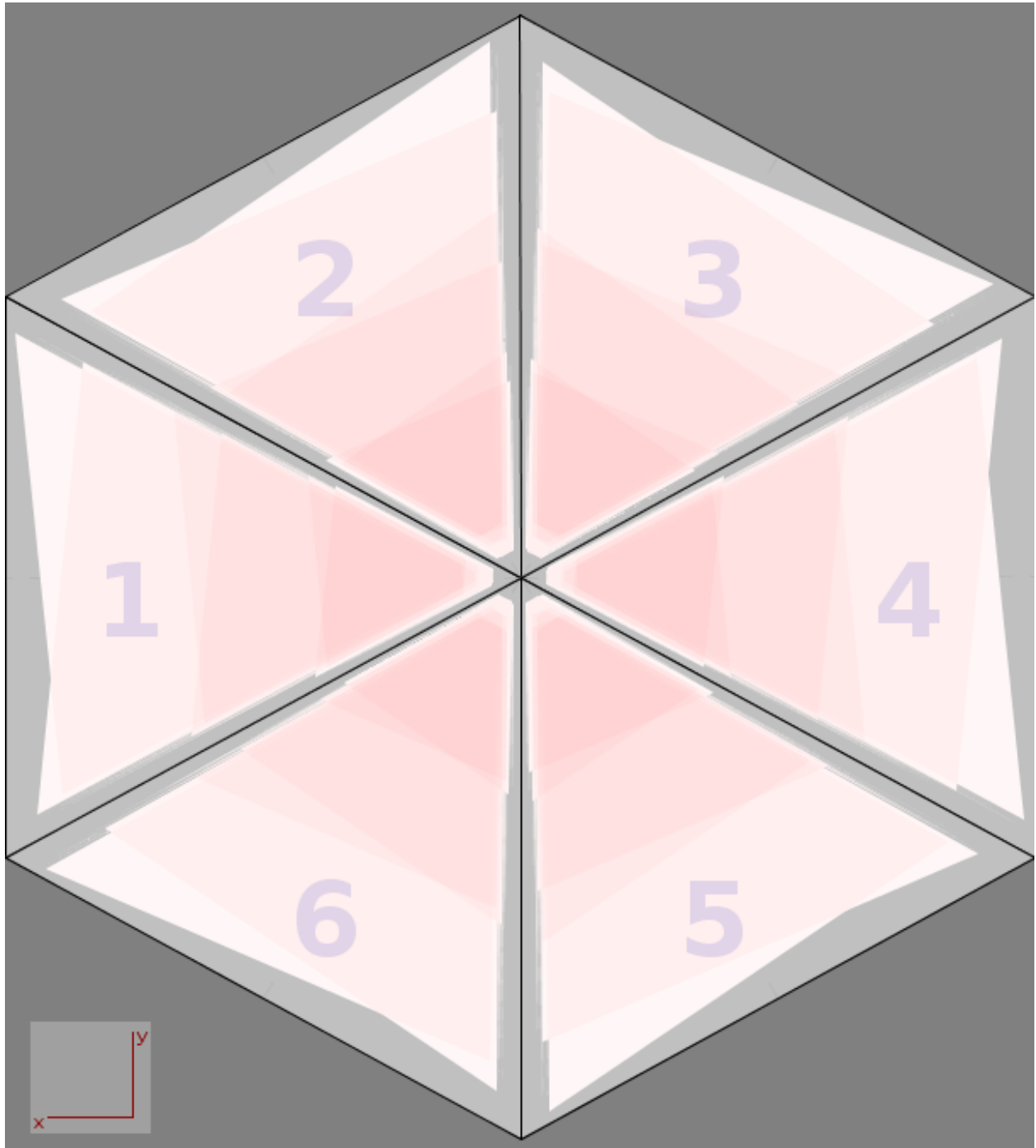


Figure 2.3: The lab coordinate system and sectors as seen when looking down the beam line.

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta) & -\sin(\theta) \\ 0 & \sin(\theta) & \cos(\theta) \end{bmatrix} \quad (2.4)$$

It follows then that to transform from the tilted sector coordinate system to the sector coordinate system, one would use matrix 2.3 with  $\theta = 25^\circ$ . Likewise, to transform from the sector coordinate system for sector  $n$  to the lab coordinate system, one would use matrix 2.4 with  $\theta = (n - 1)30^\circ$ .

### 2.2.2 Runge-Kutta Approximation

The Runge-Kutta approximations are a family of numerical methods used to approximate solutions to ordinary differential equations. The simplest member of this family is Euler's method as shown in equations 2.5 and 2.6. Here it is important to see that  $f(t, y) = \frac{df}{dt}$  and  $h$  is the step size in  $t$ . It is straightforward to see that it is possible to achieve an arbitrary level of accuracy by allowing  $h$  to be arbitrarily small.<sup>12</sup>

$$x_{n+1} = x_n + hf(t_n, y_n) \quad (2.5)$$

$$t_{n+1} = t_n + h \quad (2.6)$$

However, it is often either not possible or not feasible to allow  $h$  to become arbitrarily small. This has motivated the use of the higher order Runge-Kutta methods as they are more algorithmically efficient at giving approximations of the same quality. This allows for larger step sizes and thus fewer steps. Of these methods, the 4th order method, often referred to simply as "the Runge-Kutta method," is used in the drift chamber reconstruction process. At its core, the 4th order method is a weighted average of 4 different approximations as seen in equations 2.7 through 2.12.<sup>12</sup>

$$k_1 = f(t_n, x_n) \quad (2.7)$$

$$k_2 = f\left(t_n + \frac{h}{2}, x_n + h\frac{k_1}{2}\right) \quad (2.8)$$

$$k_3 = f(t_n + \frac{h}{2}, x_n + h\frac{k_2}{2}) \quad (2.9)$$

$$k_4 = f(t_n + h, x_n + hk_3) \quad (2.10)$$

$$x_{n+1} = x_n + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4) \quad (2.11)$$

$$t_{n+1} = t_n + h \quad (2.12)$$

It is important to note here that all members of the Runge-Kutta family do accumulate errors as they step through more iterations. The error in a single step, called the Local Truncation Error, is of the order  $O(h^5)$ , and the accumulated error is of the order  $O(h^4)$ .<sup>12</sup>

Jefferson Lab's software team has implemented the 4th order algorithm into an object called a Swimmer. This Swimmer is given maps of the magnetic fields and a state vector which includes the position, momentum, and charge of the particle of interest. The swimmer can then be called upon to swim (step) the particle until it meets certain position criteria such as a particular value along the z-axis.

### 2.2.3 Kalman Fitter

The Kalman Fitter is built on the concept of a Kalman Filter. A Kalman Filter is a statistical algorithm which combines a hypothetical model with measurements to find the Best Linear Unbiased Estimate (BLUE) of an evolved system. Included in this section is a description of the Kalman Filter process, a summary of the mathematics of it, and a discussion of the implementation used in the Hit Based and Time Based Reconstruction algorithms.

In general, the Kalman Filter Algorithm proceeds as follows. First, the initial state vector of interest and its associated uncertainties are evolved until there is a measurement available to compare against. The algorithm then compares the predicted and measured values and takes a weighted average of them wherein more weight is given to values with smaller uncertainties. The state vector is updated and the algorithm repeats. It is worth mentioning that the Kalman Filter is efficient in the sense that there is no need to keep track of old state vectors. There is instead a covariance matrix which is updated every step.<sup>13</sup>

A summary of the mathematics of the Kalman Filter is layed out in equations 2.13 through 2.28. It is quoted from *An Introduction to Kalman Filtering with Applications* by Miller and Leskiw.<sup>13</sup>

Let  $\mathbf{x}(t)$  be a state vector which satisfies the deterministic ordinary differential system

$$\dot{\mathbf{x}}(t) = F(t)\mathbf{x}(t) \quad (2.13)$$

$$\mathbf{x}(t_0) = \mathbf{x}_0 \quad (2.14)$$

Then in the Kalman theory we have:

#### System model

$$\dot{\mathbf{X}}(t) = F(t)\mathbf{X}(t) + \mathbf{V}(t) \quad (2.15)$$

$$\mathbf{X}(t_0) = \hat{\mathbf{x}}_0 \quad (2.16)$$

where  $\mathbf{V}(t)$  has mean zero and cross-covariance matrix  $S(t, s) = B(t)\Omega(t)B'(t)\delta(t - s)$  where  $\Omega(t)$  is a positive definite spectral density matrix and  $B(t)$  is a square matrix. The random initial condition  $\hat{\mathbf{x}}_0$  has mean  $\mathbf{x}_0$  and positive definite covariance matrix  $P_0$ .

#### Measurement model

$$\mathbf{Z}_k = H_k\mathbf{x}(t_k) + \mathbf{v}_k, k = 1, 2, \dots \quad (2.17)$$

where  $\mathbf{v}_k$  has mean zero and positive definite covariance matrix  $R_k$ . It is assumed that  $\mathbf{V}(t)$ ,  $\hat{\mathbf{x}}_0$ , and  $\mathbf{v}_k, k = 1, 2, \dots$  are independent.

#### State estimate propagation

$$\dot{\hat{\mathbf{x}}}(t) = F(t)\hat{\mathbf{x}}(t), t_0 \leq t < t_l \quad (2.18)$$

$$\hat{\mathbf{x}}(t_0) = \hat{\mathbf{x}}_0 \quad (2.19)$$

and

$$\dot{\hat{\mathbf{x}}} = F(t)\hat{\mathbf{x}}(t) \quad (2.20)$$

has initial condition  $\hat{\mathbf{x}}(t_k)$  in  $[t_k, t_{k+1}, \dots), k = 1, 2, \dots$

### Error covariance propagation

$$\dot{P}(t) = F(t)P(t) + P(t)F'(t) + B(t)\Omega(t)B'(t), t_0 \leq t < t_l \quad (2.21)$$

$$P(t_0) = P_0 \quad (2.22)$$

and

$$\dot{P}(t) = F(t)P(t) + P(t)F'(t) + B(t)\Omega(t)B'(t) \quad (2.23)$$

has initial condition  $P(t_k)$  in  $[t_k, t_{k+1}, \dots)$ ,  $k = 1, 2, \dots$

### State estimate update

$$\hat{\mathbf{x}}(t_k) = (I - K_k H_k) \hat{\mathbf{x}}(t_k^-) + K_k \mathbf{Z}_k \quad (2.24)$$

$$= P(t_k)[P^{-1}(t_k^-) \hat{\mathbf{x}}(t_k^-) + H'_k R_k^{-1} \mathbf{Z}_k], k = 1, 2, \dots \quad (2.25)$$

### Error covariance update

$$P(t_k) = [P^{-1}(t_k^-) + H'_k R_k^{-1} H_k]^{-1} \quad (2.26)$$

$$= (I - K_k H_k) P(t_k^-), k = 1, 2, \dots \quad (2.27)$$

### Gain matrix

$$K_k = P(t_k) H'_k R_k^{-1}, k = 1, 2, \dots \quad (2.28)$$

This is implemented in the Fitter as a method of refining a trajectory provided by calculations done on the raw hits. In essence, this method takes the initial state vector and steps it to each drift chamber region using the Runge-Kutta based Swimmer discussed in the previous section. At each region, the Kalman Filter is used to refine the trajectory. After finishing with the third region, the algorithm then steps back to near the starting location of the initial state vector and repeats the process up to 30 times. The algorithm will exit early if the  $\chi^2$  is sufficiently small.

## 2.3 Rastered Beam

A rastered beam is the intentional movement of the incident electrons, ideally translating without rotation, over the surface of the target. The primary purpose of this is to distribute the energy deposited into the target more uniformly and avoid localized target heating that would result in physical changes to the target. In the case of a cryogenic target, overheating can cause localized target boiling, reducing the effective length of the target, and result in a lower luminosity of the experiment. In the case of a polarized target, overheating can impact the ability of the target to be polarized resulting in a need to recover target polarization through annealing.

As already stated, the ideal case is a raster that purely translates the incident electrons in the plane perpendicular to the electron momentum without any rotation. The underlying assumption used to determine the kinematics of an observed scattering event is that the incident electrons were directed parallel to a coordinate axis. In the CLAS12, the reference coordinate system aligns the z-axis such that it points parallel to the incident electron's momentum. The scattered electron's azimuthal angle with respect to this axis is used to quantify kinematic variables in the reaction and the cross section. Most notably, this angle is used to determine the amount of four-momentum that was transferred to the target. It follows that an angular offset in the incident electron beam would be an error that propagates to the measured kinematics and cross-sections.

Unfortunately, it is nigh impossible to be able to achieve pure translation in a real lab setting. It is preferable then to simply minimize the changes in incident angle to such a degree that the contribution to systematic errors are minimal. In the common case of using a dipole magnet to create raster behavior, it is sufficient that the angular size of the target relative to the magnet's position be quite small. This ensures that any particle that hits the target will also have a small relative change in angle compared to any other particle.

# Chapter 3

## Simulation and Reconstruction

This chapter describes the steps taken to determine the efficacy of the proposed reconstruction algorithm. Included are a description of the physical setup of the actual polarized SIDIS dual target, a description of the simulations done to gather test data, and a discussion of both the current and proposed reconstruction algorithms. The ultimate goal of the proposed reconstruction algorithm is to be able to confidently state which of the two subtargets any given event spawned from.

### 3.1 Physical Setup

Figure 3.1 provides a general description of the target used in the simulation. In the simulation, the setup is centered in the middle of the solenoid magnet with a gap of four centimeters between the two targets. Each target is two centimeters long and has a radius of one centimeter. Each of the targets can be independently polarized and may contain either  $\text{NH}_3$  or  $\text{ND}_3$ . Additionally, there is a carbon foil centered between the two targets. This foil is used to normalize away particles scattered off of the nitrogen in the two main targets.



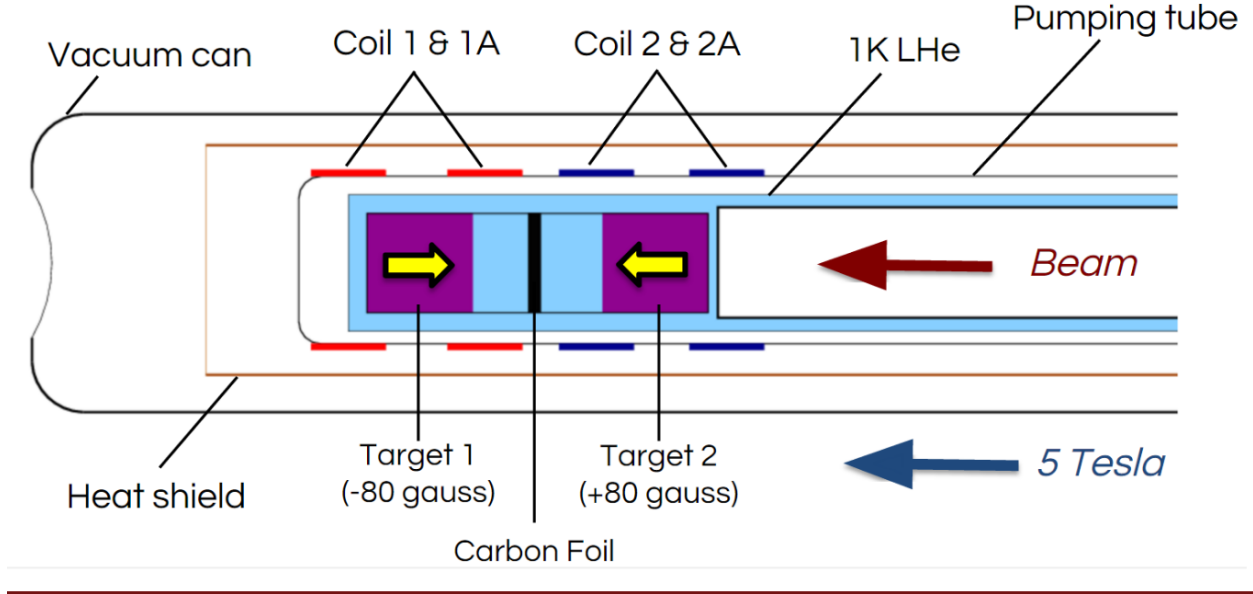


Figure 3.1: A description of the general geometry of the polarized dual target.<sup>14</sup>

## 3.2 Simulation

All of the data used to analyze the efficacy of the proposed reconstruction algorithm was generated using GEMC version 4a.2.4. GEMC (GEant4 Monte-Carlo)<sup>2</sup> takes as input configuration files called GCards. These GCards direct the program to load in certain tagged releases of the files (geometry, magnetic fields, *etc.*) required to simulate the desired experiment. The CLAS12 software team has created a repository containing various different revisions of tags which can be used.<sup>15</sup> A sample GCard can be found in Appendix A.

Additionally, the GCards allow the user to set certain parameters within the simulation. Of particular importance are scaling the magnetic field vectors and specifying the manner in which the simulated particles are generated. The magnetic field scales are set by simply passing in the desired scale factor. The event generator is somewhat more complicated. It is broken into four different components: beam momentum, momentum spread, beam position, and position spread. The beam momentum has four parameters: simulated particle, momentum magnitude, momentum polar angle, and momentum azimuthal angle. The momentum spread has three parameters: plus or minus momentum spread, plus or minus polar angle,

and plus or minus azimuthal angle. The beam position has three parameters: X-position, Y-position, and Z-position. the position spread has two parameters: spread radius in the X-Y plane and Z-position spread. A full description of all available options can be found at <https://gemc.jlab.org/gemc/html/documentation/options.html>.

It should be noted here that the ranges for the momentum were provided by Dr. Tony Forest and that they are indicative of the expected kinematic range of the actual experiment.

	e- In Up	e- In Down	e- Out Up	e- Out Down
Toroid Scale	-1.0	-1.0	1.0	1.0
$ P $	$4.75 \pm 3.25 GeV$	$4.75 \pm 3.25 GeV$	$4.75 \pm 3.25 GeV$	$4.75 \pm 3.25 GeV$
$\phi$	$22.5 \pm 17.5^\circ$	$22.5 \pm 17.5^\circ$	$22.5 \pm 17.5^\circ$	$22.5 \pm 17.5^\circ$
$\theta$	$0.0 \pm 180.0^\circ$	$0.0 \pm 180.0^\circ$	$0.0 \pm 180.0^\circ$	$0.0 \pm 180.0^\circ$
$X$	$0.0 cm$	$0.0 cm$	$0.0 cm$	$0.0 cm$
$Y$	$0.0 cm$	$0.0 cm$	$0.0 cm$	$0.0 cm$
$R$	$1.0 cm$	$1.0 cm$	$1.0 cm$	$1.0 cm$
$Z$	$-3.0 \pm 1.0 cm$	$3.0 \pm 1.0 cm$	$-3.0 \pm 1.0 cm$	$3.0 \pm 1.0 cm$
	$\pi^+$ Up	$\pi^+$ Down	$\pi^-$ Up	$\pi^-$ Down
Toroid Scale	1.0	1.0	-1.0	-1.0
$ P $	$1.8 \pm 1.0 GeV$	$1.8 \pm 1.0 GeV$	$1.8 \pm 1.0 GeV$	$1.8 \pm 1.0 GeV$
$\phi$	$45.0 \pm 45.0^\circ$	$45.0 \pm 45.0^\circ$	$45.0 \pm 45.0^\circ$	$45.0 \pm 45.0^\circ$
$\theta$	$0.0 \pm 180.0^\circ$	$0.0 \pm 180.0^\circ$	$0.0 \pm 180.0^\circ$	$0.0 \pm 180.0^\circ$
$X$	$0.0 cm$	$0.0 cm$	$0.0 cm$	$0.0 cm$
$Y$	$0.0 cm$	$0.0 cm$	$0.0 cm$	$0.0 cm$
$R$	$1.0 cm$	$1.0 cm$	$1.0 cm$	$1.0 cm$
$Z$	$-3.0 \pm 1.0 cm$	$3.0 \pm 1.0 cm$	$-3.0 \pm 1.0 cm$	$3.0 \pm 1.0 cm$

Table 3.1: Descriptions of the non-default parameters used generate the eight data sets. Inbending and outbending are shortened to In and Out respectively. Upstream and Downstream are shortened to Up and Down respectively.

Four pairs of data sets were generated for this analysis. Each pair consists of one 5000 event data set generated uniformly within each subtarget and was later combined to make a single 10000 event data set. The four ultimate data sets are labeled as such: electron-inbending, electron-outbending, positive pion, and negative pion. The toroidal field was set such the the pions are always inbending. The two electron set parameters differ only by the scale of the toroid. In one case it is set such that electrons bend into the beam line and in the other case they bend away. In the case of the pions, the toroid scale is set such that particles always bend into the beamline. Table 3.1 shows all of the non-default parameters used to generate each data set.

After GEMC has been run, the resulting EVIO (Event Input Output) files must be converted to HIPO (High Performance Output) files. This is done by calling one of the utilities packaged in clas12-offline-software, `evio2hipo`. Once that is done, the upstream and downstream components of each pair of data sets are merged using another clas12-offline-software program, `hipo-utils`. Examples for running all of these can be found in Appendix A. All that remains then is to run the HIPO files through the two pre-existing reconstruction algorithms and then the proposed algorithm.

### 3.3 Current Reconstruction Algorithms

There are currently two drift chamber reconstruction algorithms. The first, called Hit Based, feeds into the second, called Time Based. The following paragraphs are brief descriptions of both of them. Appendix B holds lengthy descriptions of both algorithms.

The Hit Based algorithm is the first pass at reconstruction and is largely used to lay groundwork for the Time Based algorithm. In broad strokes, the Hit Based algorithm works by first collecting all of the drift chamber hits and grouping them by their superlayer. Those groups are then processed and a linear fit is attempted. That fit is then used to find a position-momentum state vector between the superlayers of each region. Finally, a combination of a Kalman Fitter and the Runge-Kutta based Swimmer are used to reconstruct the

trajectory of the particle and, thus, the interaction vertex.

The Time Based algorithm works by taking the tracks already reconstructed by the Hit Based reconstruction and refining them further. First, the groups of hits are regathered and some of the likely-extraneous hits are culled using timing information. Those groups then undergo another test to check whether or not the entire group should be thrown out or not. The Kalman-Fitter and Swimmer are then used again to find the new trajectories and interaction vertices.

Both of these reconstruction algorithms are run on each HIPO file by calling the `notsouseful-util` program that is packaged as a part of `clas12-offline-software`. Once again, an example of the command to call the program can be found in Appendix A.

### 3.4 Proposed Reconstruction Algorithm

The Raster Based algorithm takes the output of both the Time and Hit Based algorithms and attempts to fit them to an off-axis beam position. This is achieved by performing a grid search through a small region of momentum space about a state vector provided by older algorithms. The actual vertex is found by swimming all of these slightly varied state vectors to the beam line and calculating the DOCA. Please note that the source code for this is found in its entirety in Appendix C.

This program currently only exists in a fork off of Jefferson Lab’s official GitHub repository. The address for this is <https://github.com/friadavi/clas12-offline-software>. Should the reader wish to run the program, they may run the `fritracking` program which is packaged with the rest of the `clas12-offline-software` in the aforementioned fork. An example of calling the program may be found in Appendix A.

For this analysis, the data sets reconstructed by the Hit and Time based algorithms were reconstructed multiple times while the two grid search parameters, `span` and `samples`, were varied independently. The `span` was varied, inclusively, between values of 0.001 and 0.015 (0.1 and 1.5%). The number of samples was varied, again inclusively, between 3 and 10.

This allows for an analysis of the reconstruction as a function of the two independently.

### 3.4.1 Parameters

This algorithm takes a variety of input parameters which control both the extent and granularity of the two searches involved as well as various other variables. These parameters are listed and explained below.

**Input** The name and path to the input file to reconstruct.

**Output** The name and path of the output file. Any preexisting file will be overwritten.

**Engine** The output from which engine to reconstruct. The options at present are Hit Based and Time Based. In the event that this parameter is not specified, the algorithm will preferentially take the Time Based if available, and Hit Based if not.

**Solenoid** The scale factor of the magnetic field generated by the solenoid.

**Toroid** The scale factor of the magnetic field generated by the toroid.

**Grid Search Span** The span in each Cartesian dimension in momentum space about which the grid search will take place. This value is provided as a percentage such that the span of the search is  $p_{x,y,z}(1 \pm \text{value})$ .

**Grid Search Samples** The number of evenly spaced sample points to take in each momentum dimension. This includes the edges of the span, as such this number should never be less than two. It is recommended to always use an odd number, that way the starting value is included in the search.

**Vertex Search Upper Bound** The upper bound in the z-direction in which to search for the DOCA to the beam position.

**Vertex Search Lower Bound** The lower bound in the z-direction in which to search for the DOCA to the beam position.

**Vertex Samples** The number of evenly spaced sample points to take in each iteration of the vertex search. This includes upper and lower bounds, as such this number should not be less than two.

**Vertex Iterations** The number of recursive iterations to go through in the vertex search.

**Threads** The maximum number of threads that can be used in the reconstruction.

### 3.4.2 Overarching Algorithm

The algorithm begins by parsing commandline options; it stops the reconstruction if any errors occur. After that, the magnetic field maps are initialized and scaled if necessary. Then, for each event within the input file, the reconstruction algorithm is run and the output, if any, is written to the output file.

### 3.4.3 Reconstruction

This begins by reading in whichever output banks are required, as dictated by the Engine commandline parameter. Then the beam position is read in from the simulation bank. If the event is real data, then the beam position is assumed to be at  $x = 0, y = 0$ . There is currently an assumed radius of error of 0.5mm in the beam position. The algorithm then collects a state vector output by the Hit Based and Time Based algorithm. This state vector's position is just upstream of the region 1 drift chamber and will be the starting point of the search. The grid search is run and the output is written to the data banks.

### 3.4.4 Grid Search

This sub-algorithm starts by computing the upper and lower bounds of the momentum space in which to search. Arrays to store the output of each search point are then created and a ThreadPool is established to allow for the multithreading of the vertex searches for each grid search sample point. A vertex search job thread is then submitted for every sample

point in the three dimensional momentum search space. For the Grid Search Samples value  $n$ , there are  $n^3$  sample points, each evenly spaced in all three dimensions.

After all submitted jobs have been completed, the algorithm searches through all of the outputs and identifies interaction vertices which are either within the beam line's margin of error or a local minimum. All other points are culled. Any points whose interaction vertex  $z$  value is outside of the vertex search bounds is culled. Finally, the algorithm searches through all surviving interaction vertices and selects for output the one whose initial starting momentum deviated the least from the Hit Based or Time Based momentum.

The source code for this method starts on line 611 of Appendix C.

### 3.4.5 Vertex Search

This sub-algorithm begins by swimming from the state vector provided by the grid search to each sample  $z$  value within the search region. The samples points are evenly spaced throughout the region and include the boundaries. The swim outputs are collected, and the DOCA for each one is calculated. All local minima among the DOCAs are added to a list. If this is the last iteration, the algorithm returns the output which corresponds to the smallest DOCA, failing gracefully if no swims performed successfully.

In the event that there are more iterations to go, this algorithm recursively calls itself on each local minima. The bounds of the recursive search search are as follows: if the minimum is at a boundary of the search region it is recentered on the boundary without changing the actual distance spanned and the function is called without decrementing the iteration count. Otherwise, the search region is recentered on the point associated with the local minimum and the boundaries are shrunk to coincide with the nearest neighbor sample points. The iteration count is decremented and the function is called. The output of each new search is collected and the one with the smallest DOCA is returned.

The source code for this method starts on line 782 of Appendix C.

# Chapter 4

## Analysis

Before moving into the analysis proper, it is worth summarizing what exactly is being analyzed and to what end. Four data sets, each initially of 10,000 events, have been generated and run through the Hit Based and Time Based reconstruction algorithms. The outputs of the old reconstructions were then fed into the proposed reconstruction multiple times while varying the number of samples used and the span of the search. These data sets consist of events that were generated within the volume of the subtargets of the polarized SIDIS dual target. Two of the data sets consist of electrons, one with an in-bending toroid configuration and one with an out-bending configuration. The other two data sets are  $\pi^+$  and  $\pi^-$  particles, each with a toroid configuration which will cause them to bend into the beam line. The goal is to determine with how much confidence one can state which of the targets a particle was generated from.

The analysis will be presented as follows. First a comparison of typical  $\Delta Z$  values will be shown. Then an examination of the effect of the raster radius on the reconstruction will be provided. That will be followed by an examination of typical  $\Delta X$  and  $\Delta Y$  reconstructions. Finally, a discussion of the reconstruction efficiency will be presented.



## 4.1 $\Delta Z$ Comparisons

Figures 4.1 through 4.4 show histograms and associated Gaussian fits of the differences between the generated Z positions and the reconstructed Z positions for both the current and proposed reconstruction algorithms. It is immediately obvious that the electrons are reconstructed more accurately and precisely than the pions. Furthermore, it is worth looking at the number of entries in each figure. They indicate that it is perhaps unwise to put too much weight in the pion reconstructions due to likely insufficient statistics. Regardless, the  $\sigma$  of each fit to the proposed algorithm data shows an improvement over the current algorithm. Commentary on what can be said for confidence in determining which target any given event came from is reserved for the Conclusion section.

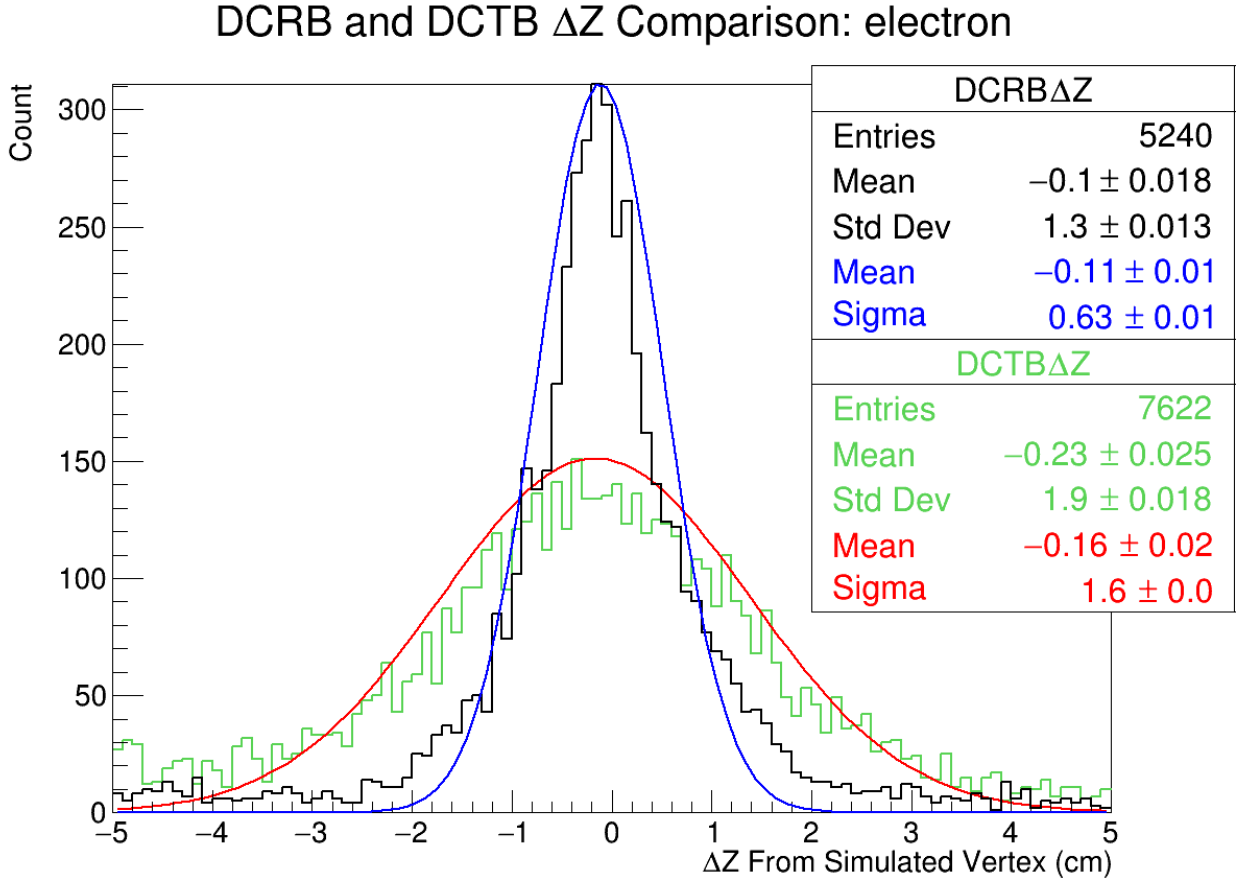


Figure 4.1: The difference between the generated and reconstructed Z positions of the interaction vertex for the HB/TB algorithm as well as the Raster Based for in-bending electrons.

$Span = 0.005, Samples = 7$

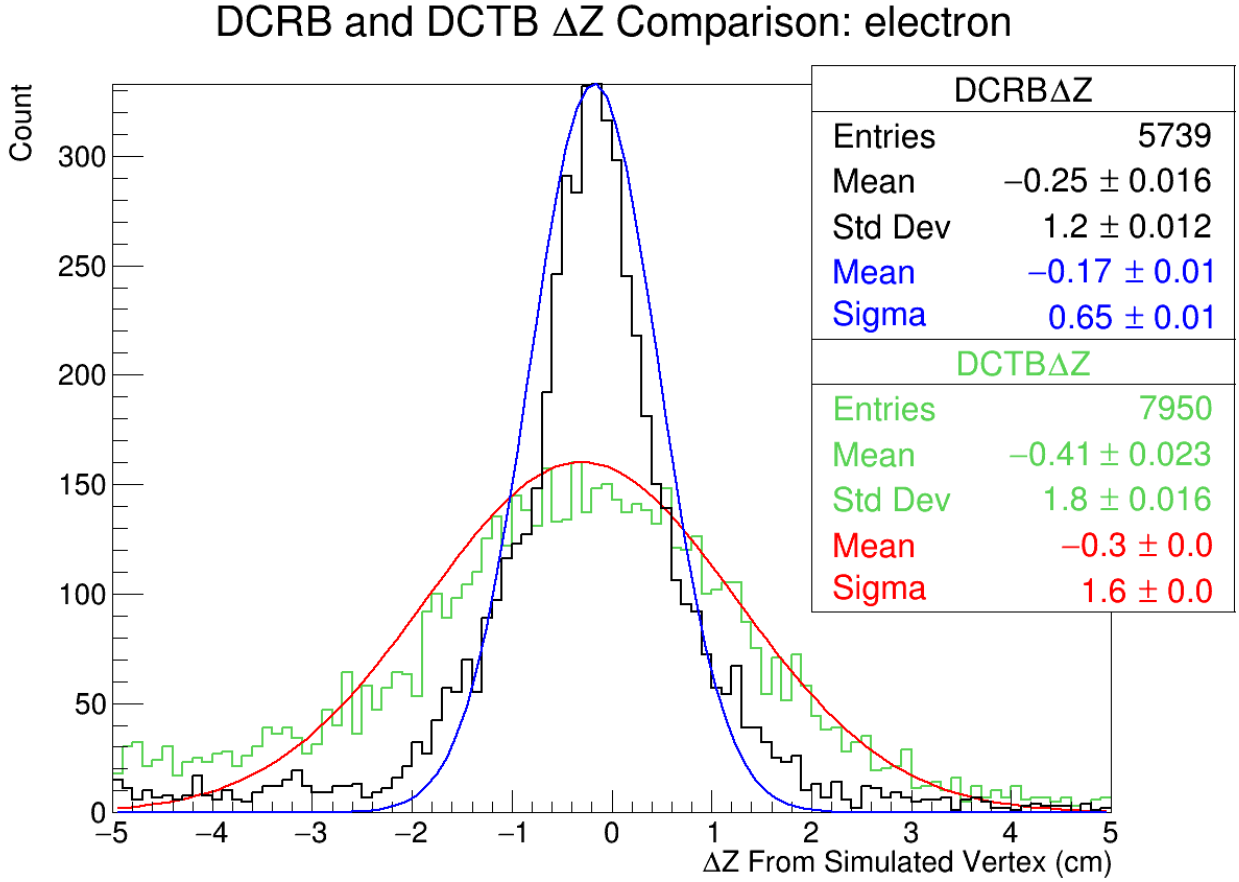


Figure 4.2: The difference between the generated and reconstructed Z positions of the interaction vertex for the HB/TB algorithm as well as the Raster Based for out-bending electrons.

$Span = 0.005, Samples = 7$

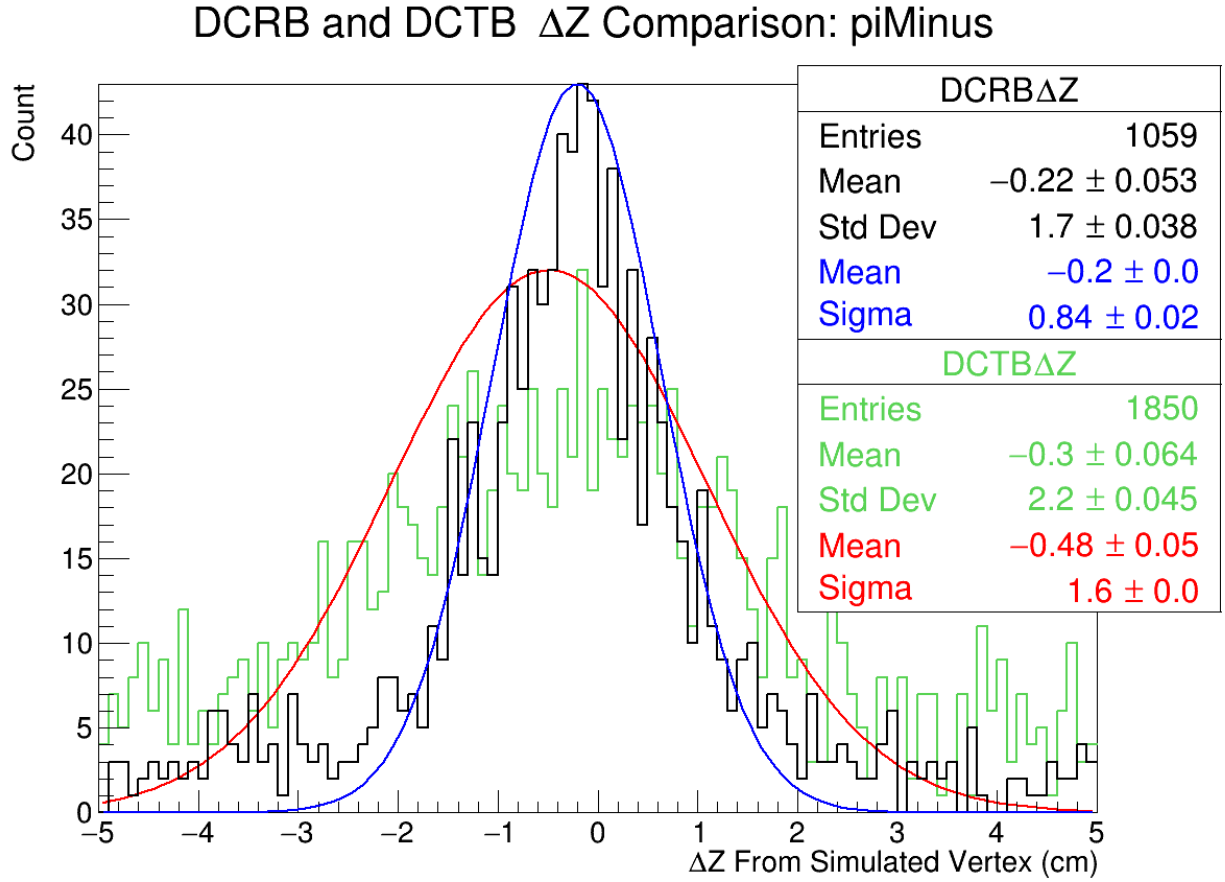


Figure 4.3: The difference between the generated and reconstructed Z positions of the interaction vertex for the HB/TB algorithm as well as the Raster Based for in-bending  $\pi^-$ s.  
 $Span = 0.005, Samples = 7$

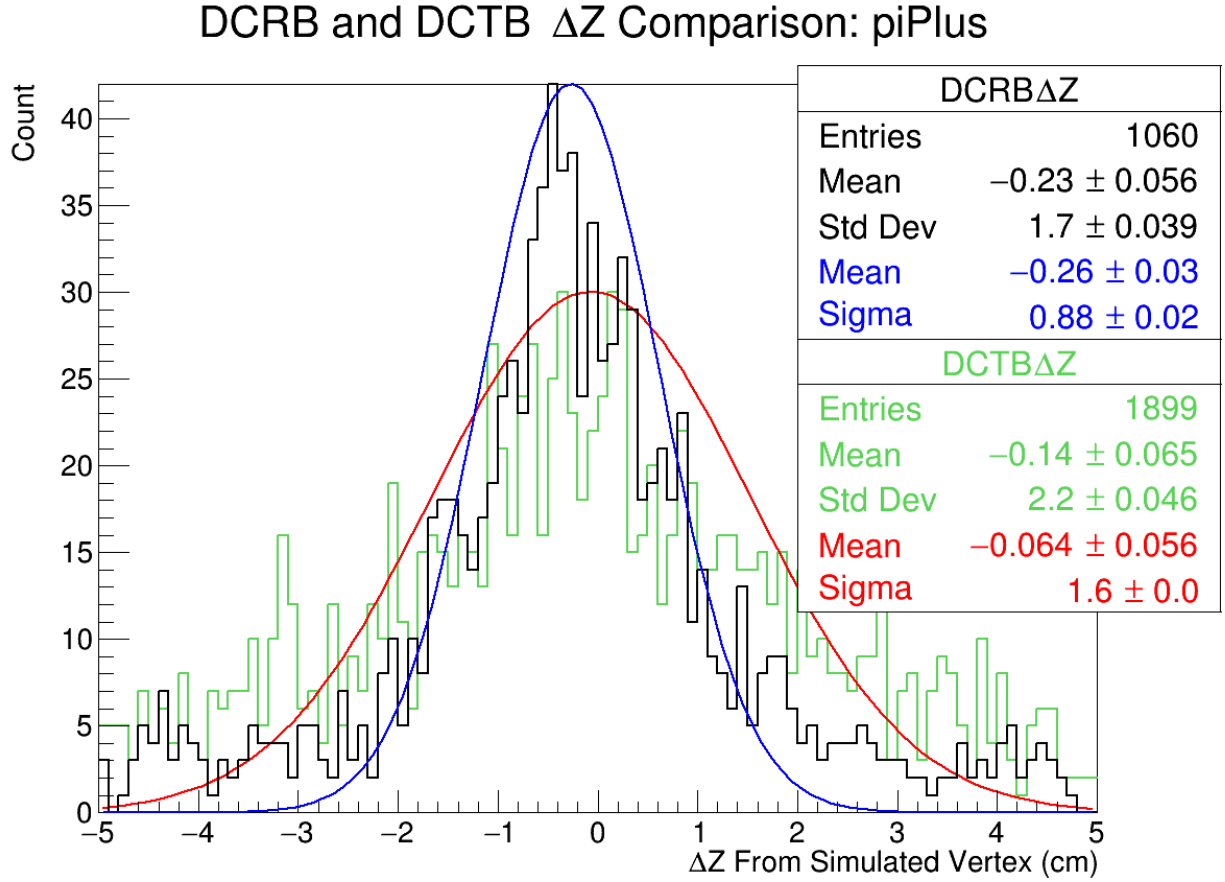


Figure 4.4: The difference between the generated and reconstructed Z positions of the interaction vertex for the HB/TB algorithm as well as the Raster Based for in-bending  $\pi^+$ s.  $Span = 0.005, Samples = 7$

## 4.2 $\Delta Z$ Summary Table

Table 4.1 summarizes the  $\Delta Z$  fit information for the varying spans for each data set. One should note the general trend towards a larger  $\sigma$  as the span grows.

Span	e- in-bending $\Delta Z$ (cm)	e- outbending $\Delta Z$ (cm)	$\pi^-$ $\Delta Z$ (cm)	$\pi^+$ $\Delta Z$ (cm)
0.001	$-0.15 \pm 0.46$	$-0.15 \pm 0.43$	$-0.28 \pm 0.71$	$-0.22 \pm 0.78$
0.002	$-0.15 \pm 0.51$	$-0.16 \pm 0.48$	$-0.28 \pm 0.73$	$-0.26 \pm 0.82$
0.003	$-0.16 \pm 0.54$	$-0.19 \pm 0.52$	$-0.26 \pm 0.72$	$-0.26 \pm 0.91$
0.004	$-0.13 \pm 0.58$	$-0.17 \pm 0.57$	$-0.26 \pm 0.79$	$-0.29 \pm 0.91$
0.005	$-0.11 \pm 0.63$	$-0.17 \pm 0.65$	$-0.20 \pm 0.84$	$-0.26 \pm 0.88$
0.006	$-0.14 \pm 0.62$	$-0.20 \pm 0.62$	$-0.25 \pm 0.83$	$-0.24 \pm 0.81$
0.007	$-0.14 \pm 0.63$	$-0.21 \pm 0.63$	$-0.25 \pm 0.78$	$-0.25 \pm 0.91$
0.008	$-0.12 \pm 0.66$	$-0.19 \pm 0.66$	$-0.23 \pm 0.73$	$-0.19 \pm 1.00$
0.009	$-0.09 \pm 0.67$	$-0.15 \pm 0.74$	$-0.21 \pm 0.83$	$-0.11 \pm 0.92$
0.010	$-0.11 \pm 0.76$	$-0.17 \pm 0.78$	$-0.20 \pm 0.87$	$-0.17 \pm 0.92$
0.011	$-0.14 \pm 0.76$	$-0.21 \pm 0.82$	$-0.22 \pm 0.90$	$-0.21 \pm 0.96$
0.012	$-0.13 \pm 0.79$	$-0.19 \pm 0.82$	$-0.16 \pm 0.91$	$-0.17 \pm 0.96$
0.013	$-0.10 \pm 0.82$	$-0.17 \pm 0.84$	$-0.15 \pm 0.91$	$-0.12 \pm 0.94$
0.014	$-0.14 \pm 0.88$	$-0.13 \pm 0.91$	$-0.14 \pm 0.82$	$-0.14 \pm 0.97$
0.015	$-0.12 \pm 0.91$	$-0.14 \pm 0.95$	$-0.18 \pm 0.95$	$-0.12 \pm 0.90$

Table 4.1: A summary of the  $\Delta Z$  fit information for the different spans of each data set.

### 4.3 $\Delta Z$ Dependence on Raster Radius

Figures 4.5 and 4.6 show the relation between the raster radius and the quality of the reconstruction in the Z direction for in-bending electrons. While the number of events is, unfortunately, small for the center the graphs still seem to suggest that the raster radius doesn't have any significant effect. The out-bending electrons behave nearly identically. The number of events in both of the pion data sets are too small to be able to say with any confidence whether or not the raster radius has any effect.

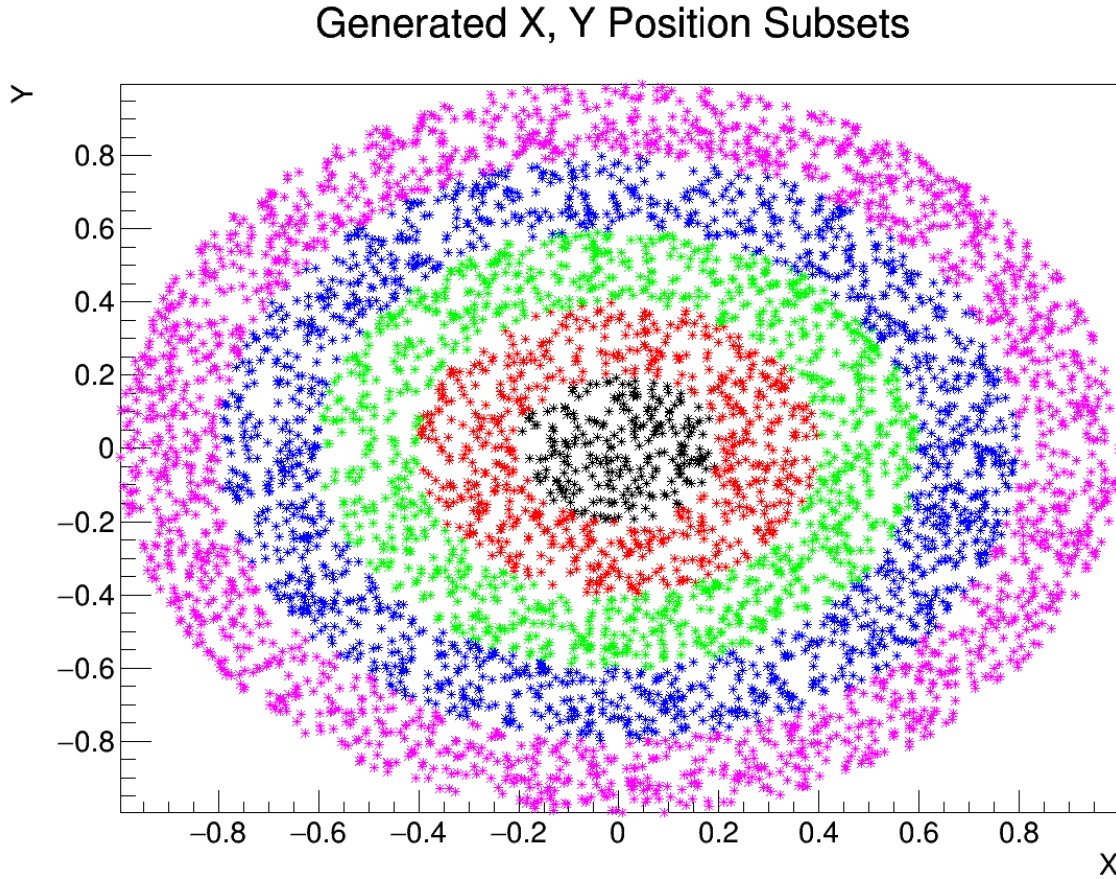


Figure 4.5: Color-coded subsets of the generated X and Y positions which correlate to in-bending electron events successfully reconstructed by the proposed algorithm.  $Span = 0.005$ ,  $Samples = 7$

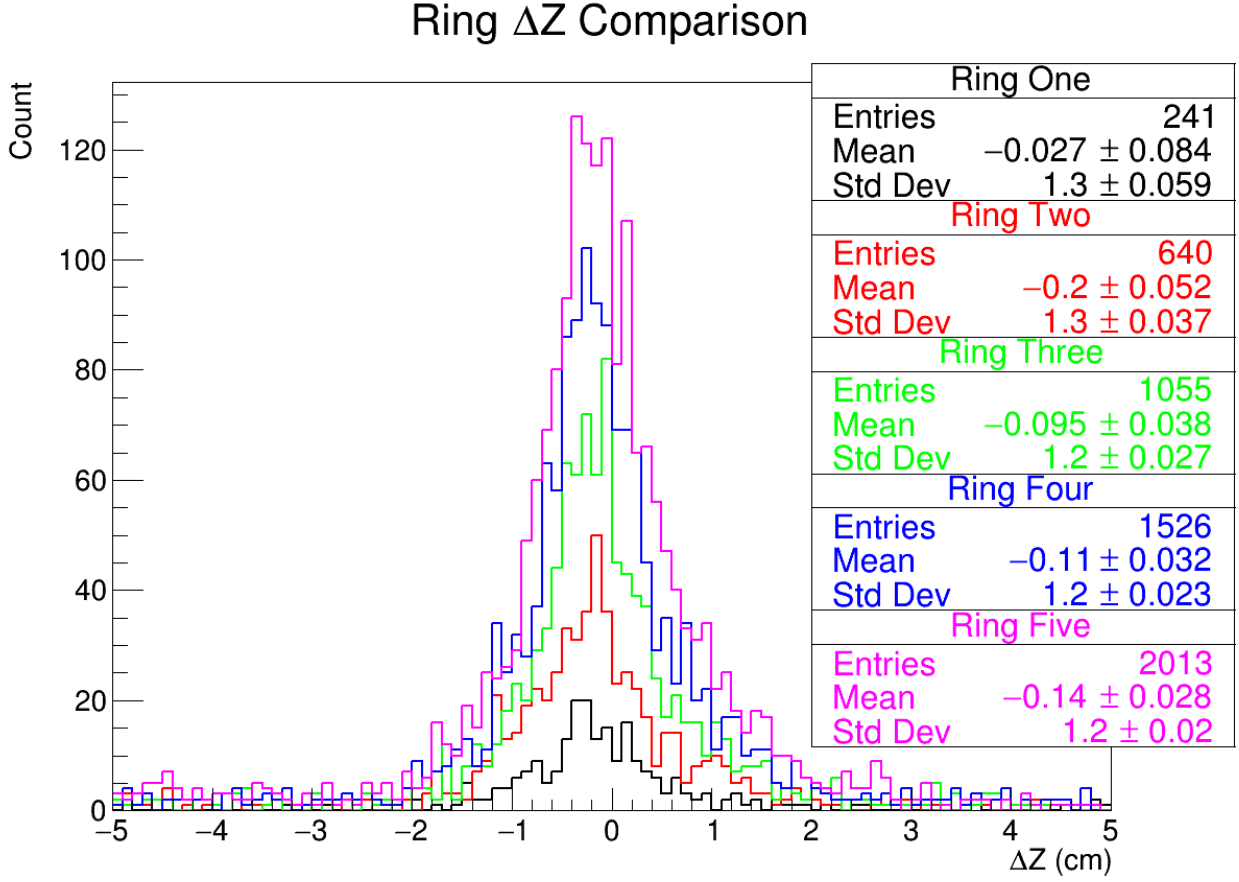


Figure 4.6:  $\Delta Z$  histogram of the subsets described by figure 4.5

## 4.4 $\Delta X$ and $\Delta Y$

Figures 4.7 and 4.8 show histograms of the  $\Delta X$  and  $\Delta Y$  which are typical of all reconstructed data sets. The improvement over the current reconstruction algorithm is obviously substantial, being an order of magnitude smaller in  $\sigma$ .



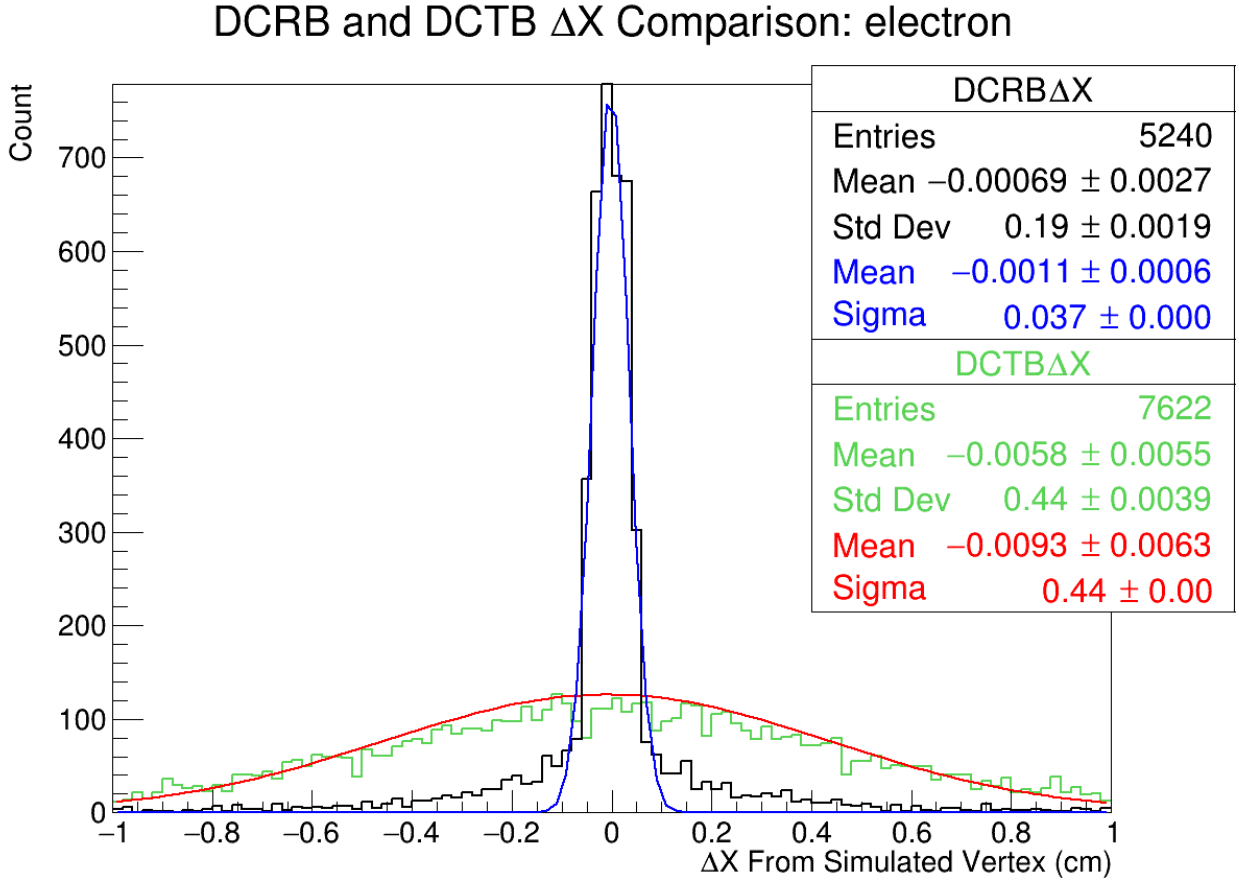


Figure 4.7: The difference between the generated and reconstructed X positions of the interaction vertex for the HB/TB algorithm as well as the Raster Based for in-bending electrons.  $Span = 0.005$ ,  $Samples = 7$

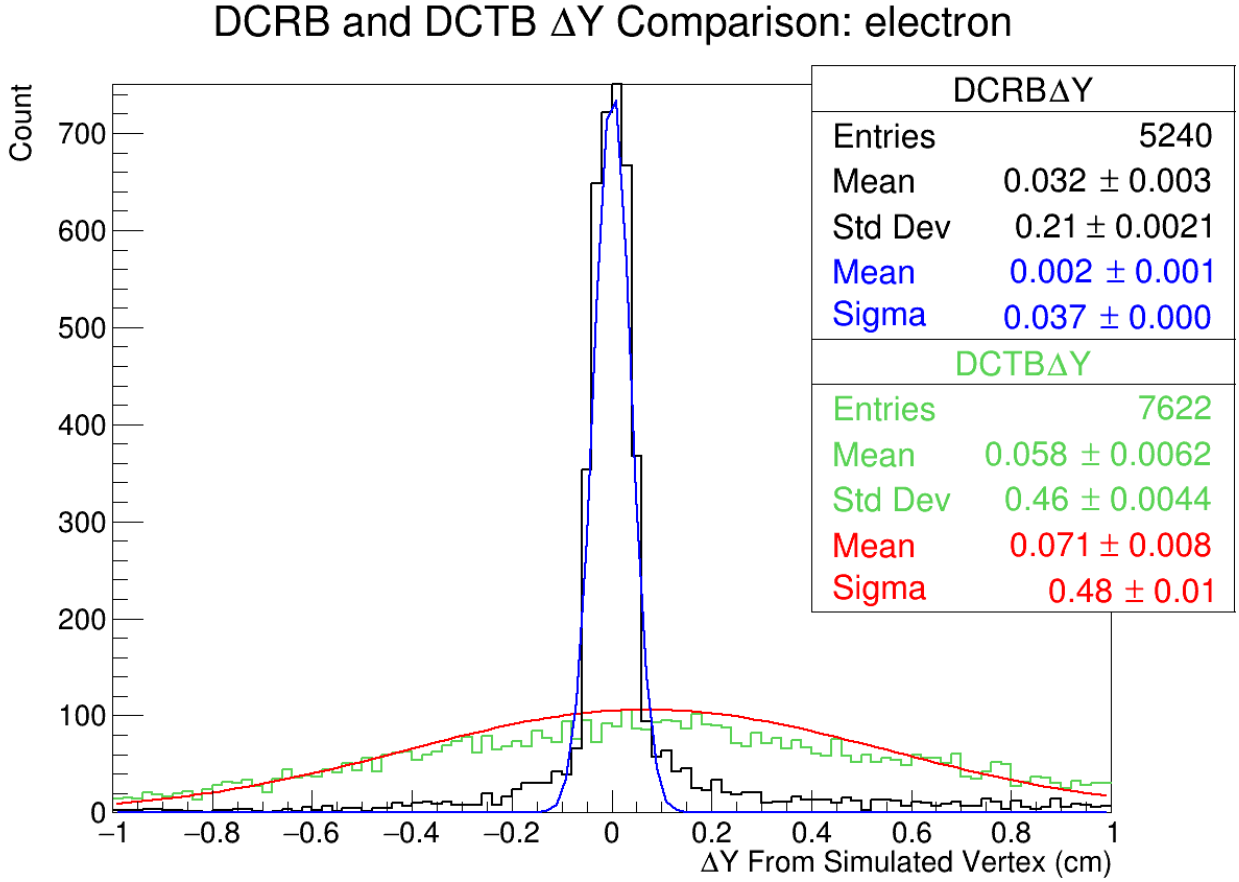


Figure 4.8: The difference between the generated and reconstructed Y positions of the interaction vertex for the HB/TB algorithm as well as the Raster Based for in-bending electrons.  $Span = 0.005$ ,  $Samples = 7$

## 4.5 Efficiency

Figure 4.9 shows the efficiency of the proposed algorithm. There is a general upwards trend in efficiency, though it should be noted that this comes at a cost in the form of a larger  $\sigma$ . This can be seen by comparing figure 4.9 to table 4.1.

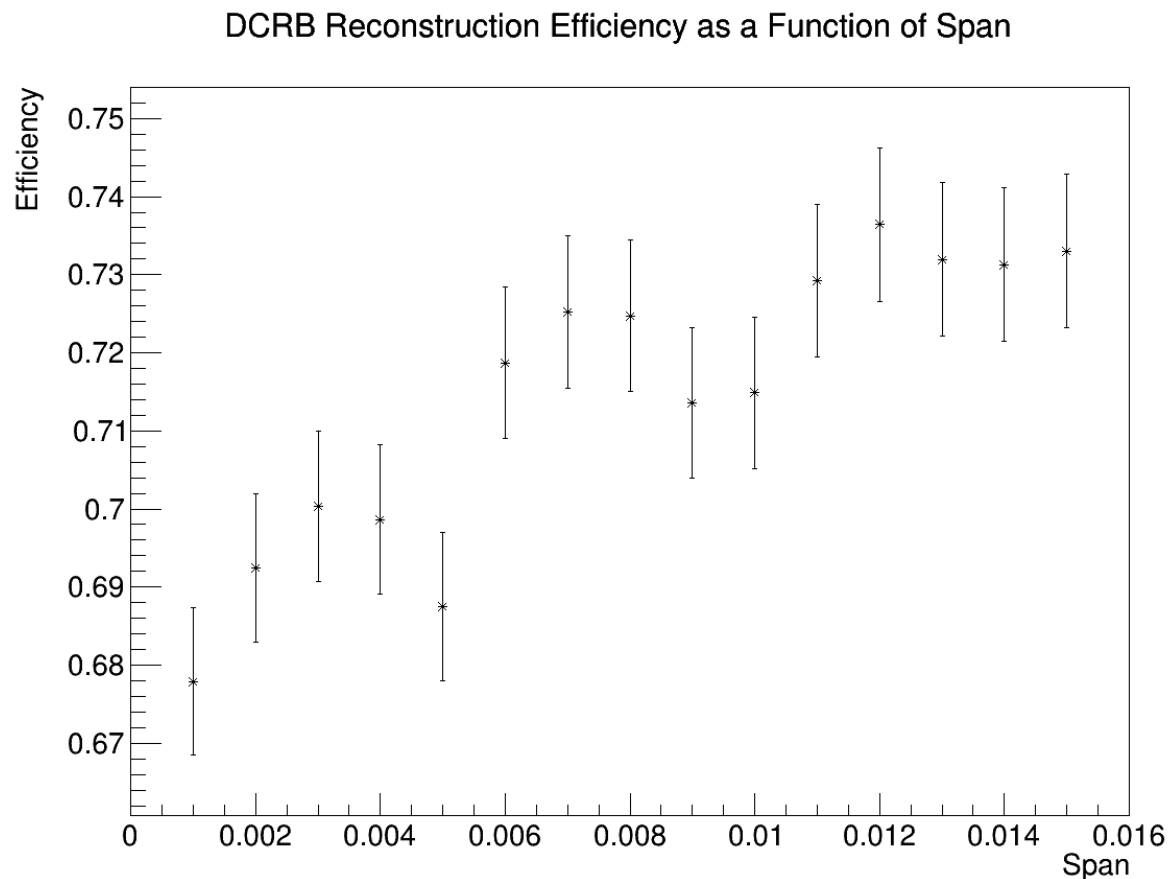


Figure 4.9: The efficiency of the proposed reconstruction algorithm for in-bending electrons. Note that this is not the total efficiency.  $Span = 0.005, Samples = 7$

# Chapter 5

## Conclusion

The goal of this project was ultimately to determine to what level of confidence one can say which of the targets, or possibly the foil, any given event came from. To that end, a  $5\sigma$  confidence level corresponds to  $8mm$  if there is no carbon foil and  $4mm$  if there is. With that stated, it is clear, by inspection of table 4.1, that one can state with  $4\sigma$  confidence which of the targets an event came from if the carbon foil is present. However, the removal of the carbon foil allows for  $5\sigma$  confidence for electron events with search spans up to 1% (0.010). Even the smallest search spans for can claim  $5\sigma$  confidence. With the carbon foil, the best that can be claimed for electrons is  $4\sigma$  and only  $2\sigma$  for pions.

It was found that the raster radius has no meaningful effect on the quality of reconstruction. Unfortunately, the span of the search does, though this could likely be mitigated by choosing more sample points. The efficiency, however does generally increase with increasing span. And lastly, the new algorithm's X and Y reconstructions are consistently an order of magnitude better than the current algorithm regardless of particle or toroidal field.

With respect to the relationship between the reconstruction efficiency and the span of the search, it would seem that the Raster Based Algorithm is acting as a filter of sorts. As the span increases, both the sigma of the reconstruction and the efficiency increases. This seems to suggest that broadening the quantity of events which can be successfully reconstructed results in a worse quality of reconstruction. This, in turn, would seem to suggest that the

algorithm intrinsically culls out events which can not be reconstructed well enough.

In the future, it would likely be worthwhile to further pursue examinations of the effects of fit parameters on the reconstruction quality and efficiency. In particular, the covariance of the number of samples and span should be thoroughly explored so as to better pick optimized parameters for each.

# References

- <sup>1</sup>Jefferson Lab, *Experimental Hall B*, <https://www.jlab.org/research/hall-b> (Accessed 12 May 2019).
- <sup>2</sup>Jefferson Lab, *GEant4 Monte-Carlo*, <https://gemc.jlab.org/gemc/html/index.html> (Accessed 02 June 2019).
- <sup>3</sup>CERN, *GEANT4 A Simulation Toolkit*, <https://geant4.web.cern.ch/> (Accessed 02 June 2019).
- <sup>4</sup>Jefferson Lab, *clas12-offline-software*, <https://github.com/JeffersonLab/clas12-offline-software> (Accessed 02 June 2019).
- <sup>5</sup>CLAS12 Drift Chamber Group, *CLAS12 Drift Chambers*, (unpublished) (2007).
- <sup>6</sup>F. Sauli, “Principles of Operation of Multiwire Proportional and Drift Chambers”, (1977).
- <sup>7</sup>W. R. Leo, *Techniques for Nuclear and Particle Physics Experiments: A How to Approach* (1987).
- <sup>8</sup>*CLAS12 - Drift Chambers (DC)*, Jefferson Lab (Mar. 2017).
- <sup>9</sup>*CLAS12 - Solenoid Magnet*, Jefferson Lab (Mar. 2017).
- <sup>10</sup>*CLAS12 - TORUS Magnet*, Jefferson Lab (Mar. 2017).
- <sup>11</sup>G. Arfken, *Mathematical methods for physicists*, Seventh (Academic Press, Inc., San Diego, 2013).
- <sup>12</sup>D. Griffiths and D. Higham, *Numerical methods for ordinary differential equations: initial value problems*, Springer Undergraduate Mathematics Series (Springer London, 2010).

<sup>13</sup>K. Miller and D. Leskiw, *An introduction to kalman filtering with applications* (Krieger Publishing Company, 1987).

<sup>14</sup>T. Forest, *Private communication*.

<sup>15</sup>Jefferson Lab, *clas12Tags*, <https://github.com/gemc/clas12Tags>.

# Appendices



# Appendix A

## Example Program Calls and Example GCard

This appendix contains examples of program calls used throughout the reconstruction process and an example GCard.

### A.1 Example GCard

Below is an example GCard that could be fed into GEMC.

```
1 <gcard>
2
3     <!-- target. Notice variation give the target type. Can be: 1H2, 1D2,
        ↪ ND3
4     <detector name="experiments/clas12/targets/cad/" factory="CAD"/>
5     <detector name="experiments/clas12/targets/target" factory="TEXT"
        ↪ variation="cad"/>
6 -->
7     <!-- PolTarg -->
8     <detector name="/jlab/workdir/Thesis/Scripts/detectors/clas12/targets/
        ↪ target" factory="TEXT" variation="PolTarg"/>
```

```

9      <detector name="/jlab/workdir/Thesis/Scripts/detectors/clas12/targets/
      ↪ PolTarg/" factory="CAD"/>
10
11     <!-- central detectors -->
12     <detector name="experiments/clas12/cnd/cnd"                factory="
      ↪ TEXT" variation="original"/>
13     <detector name="experiments/clas12/bst/bst"                factory="
      ↪ TEXT" variation="java"/>
14     <!-- detector name="experiments/clas12/bstShield/bstShield"
      ↪
      ↪ factory="TEXT" variation="w51" -->
15     <detector name="experiments/clas12/micromegas/micromegas" factory="
      ↪ TEXT" variation="michel"/>
16
17
18     <!--ctof, cad -->
19     <detector name="experiments/clas12/ctof/ctof"              factory="TEXT"
      ↪ variation="cad"/>
20     <detector name="experiments/clas12/ctof/javacad/"          factory="CAD"/>
21
22     <!--high threshold cherenkov -->
23     <detector name="experiments/clas12/htcc/htcc"              factory="TEXT"
      ↪ variation="original"/>
24
25     <!-- magnets -->
26     <detector name="experiments/clas12/magnets/solenoid"        factory="TEXT"
      ↪ variation="original"/>
27     <detector name="experiments/clas12/magnets/cad/"           factory="CAD" />
28
29
30     <!-- Beamline configuration: FT is used -->
31     <detector name="experiments/clas12/ft/ft"                  factory="
      ↪ TEXT" variation="FTOn"/>
32     <detector name="experiments/clas12/beamline/cadBeamline/"   factory="CAD
      ↪ "/>

```

```

33     <detector name="experiments/clas12/beamline/vacuumLine/"    factory="CAD
        ↪ ">
34     <detector name="experiments/clas12/beamline/beamline"      factory="
        ↪ TEXT" variation="FTOn"/>
35
36     <!-- forward carriage -->
37     <detector name="experiments/clas12/fc/forwardCarriage"    factory="TEXT"
        ↪ variation="TorusSymmetric"/>
38
39     <detector name="experiments/clas12/dc/dc"                  factory="TEXT"
        ↪ variation="java"/>
40     <detector name="experiments/clas12/ftof/ftof"              factory="TEXT"
        ↪ variation="java"/>
41     <detector name="experiments/clas12/ec/ec"                  factory="TEXT"
        ↪ variation="java"/>
42     <detector name="experiments/clas12/pcal/pcal"              factory="TEXT"
        ↪ variation="java"/>
43     <detector name="experiments/clas12/ltcc/ltcc"              factory="TEXT"
        ↪ variation="original"/>
44     <detector name="experiments/clas12/ltcc/cad_cone/"          factory="CAD"/>
45     <detector name="experiments/clas12/ltcc/cad/"              factory="CAD"/>
46
47
48     <!-- you can scale the fields here. Remember torus -1 means e-
        ↪ INBENDING -->
49     <option name="SCALE_FIELD" value="TorusSymmetric, 1.0"/>
50     <option name="SCALE_FIELD" value="clas12-newSolenoid, 1.0"/>
51
52     <!-- hall field -->
53     <option name="HALL_FIELD" value="clas12-newSolenoid"/>
54
55     <!-- fields, precise mode -->
56     <option name="FIELD_PROPERTIES" value="TorusSymmetric, 2*mm,
        ↪ G4ClassicalRK4, linear"/>

```

```

57     <option name="FIELD_PROPERTIES" value="clas12-newSolenoid, 1*mm, 1
        ↪ G4ClassicalRK4, 1linear"/>
58
59
60     <!-- beam conditions -->
61     <option name="BEAM_P" value="e-, 4.75*GeV, 22.5*deg, 0*deg"/>
62     <option name="BEAM_V" value="(0.0, 0.0, 3.0) cm"/>
63     <option name="SPREAD_P" value="3.25*GeV, 17.5*deg, 180*deg"/>
64     <option name="SPREAD_V" value="(1.0, 1.0) cm"/>
65
66
67     <option name="SAVE_ALL_MOTHERS" value="0"/>
68     <option name="RECORD_MIRRORS" value="1"/>
69
70     <option name="PHYSICS" value="FTFP_BERT, 1STD, 1Optical"/>
71
72     <option name="OUTPUT" value="evio, 1out.ev"/>
73
74     <!-- Will print message every 10 events -->
75     <option name="PRINT_EVENT" value="10" />
76
77
78     <!-- Run Number 11, picked up by digitization routines -->
79     <option name="RUNNO" value="11" />
80
81
82     <!-- RF Signal needs event time window defined by LUMI_EVENT.
83     If Backround is activated make sure to use LUMI_EVENT below instead
        ↪ .-->
84     <option name="LUMI_EVENT" value="0, 248.5*ns, 4*ns" />
85     <option name="RFSETUP" value="0.499, 40, 20" />
86
87
88

```

```

89      <!-- beam background. for 250 ns timewindow we have 124,000 e- on
90      a LH2 target at 10^35 luminosity
91      I suggest in this case to set SAVE_ALL_MOTHERS to 0
92      or the many tracks will slow down the simulation a lot
93      For background studies use field in fast mode:
94      -->
95
96      <!--
97      <option name="LUMI_EVENT"      value="124000, 248.5*ns, 4*ns" />
98      <option name="LUMI_P"          value="e-, 10.6*GeV, 0*deg, 0*deg" />
99      <option name="LUMI_V"          value="(0.0, 0.0, -10) cm" />
100     <option name="LUMI_SPREAD_V"   value="(0.03, 0.03) cm" />
101     -->
102
103     <!-- production threshold for passive volumes -->
104     <!-- beamline shielding: 10cm-->
105     <option name="PRODUCTIONCUTFORVOLUMES" value="innerShieldAndFlange,
        ↪ outerFlange, outerMount, nut1, nut2, nut3, nut4, nut5, nut6,
        ↪ nut7, nut8, nut9, taggerInnerShield, main-cone, main-cone,
        ↪ adjuster1, adjuster2, adjuster3, DSShieldFrontLead,
        ↪ DSShieldBackLead, DSShieldInnerAss, DSShieldBackAss,
        ↪ DSShieldFrontAss, DSShieldBackCover, DSShieldFrontCover,
        ↪ DSShieldFlangeAttachment, DSShieldFlange, 100" />
106
107     <!-- vacuum line: 10cm-->
108     <option name="PRODUCTIONCUTFORVOLUMES" value="
        ↪ connectUpstreamToTorusPipe, connectTorusToDownstreamPipe,
        ↪ downstreamPipeFlange, 100" />
109
110     <!-- torus magnet: 10cm-->
111     <option name="PRODUCTIONCUTFORVOLUMES" value="BoreShield, CenterTube,
        ↪ DownstreamShieldingPlate, DownstreamVacuumJacket, WarmBoreTube,
        ↪ apex, Shield1, Shield2, Shield3, Shield4, Shield5, Shield6,
        ↪ Shield7, shell1a, shell1b, shell12a, shell12b, shell13a, shell13b,

```

```

112         ↪ shell14a, _shell14b, _shell15a, _shell15b, _shell16a, _shell16b, _100" />
113
114         <!-- Commented: To remove a detector, set his existence to no. For
115             ↪ example to remove the forward micromegas.
116             <detector name="FMT">
117                 <existence exist="no" />
118             </detector>
119             -->
120
121 </gcard>

```

## A.2 Program Calls

An example of the gemc program call:

```

1 bash gemc -USE_GUI=0 -gcard=/some/path/to/input.gcard -N=10000 -DATABASE=/some
  ↪ /path/to/local/database.sqlite -OUTPUT="evio, _/some/path/to/output.evio"

```

An example of the evio2hipo program call:

```

1 bash evio2hipo -n 10000 -r 11 -s -1.0 -t -1.0 -o /some/path/to/output.hipo /
  ↪ some/path/to/input.evio

```

An example of the hipo-utils program call:

```

1 bash hipo-utils -merge -o /some/path/to/output.hipo /some/path/to/input1.hipo
  ↪ /some/path/to/input2.hipo

```

An example of the notsouseful-util program call:

```

1 bash notsouseful-util -n 10000 -c 2 -i /some/path/to/input.hipo -o /some/path/
  ↪ to/output.hipo

```

An example of the fritracking program call:

```

1 bash fritracking -s -1.0 -t -1.0 -i /some/path/to/input.hipo -o /some/path/to/
  ↪ output.hipo -x 3 -g 7 0.005 -v 9 2 -z -4.0 4.0

```

# Appendix B

## Descriptions of Hit Based and Time Based Reconstructions

This appendix contains descriptions of the previously existing drift chamber reconstruction algorithms, Hit Based and Time Based. These descriptions are current as of coatjava release 5c.7.4.

### B.1 Definition of Terms

There are several terms which must be first understood before reading further:

**DOCA** Distance of Closest Approach, i.e. the shortest distance between some point in space and some defined path.

**Hit** A representation of a measurement from an individual wire.

**Cluster** A representation of groups of physically close hits.

**Segment** A representation of a linear fit to a cluster.

**Cross** A representation of a position and momentum unit vector of the particle's trajectory at the midplane between two superlayers as reconstructed from the segments of both superlayers.

**Trajectory** A general position and momentum vector at some point in space.

**Track** A representation of a reconstructed path through the detector.

**Road** A representation of a collection of segments.

**Swimmer** A software tool utilizing a fourth order Runge-Kutta approximation to step, or "swim", a particle's trajectory through the detector.

## B.2 Hit Based Algorithm

The Hit Based algorithm takes the raw drift chamber hit positions, but not any time information beyond that which gives the position along any given wire, and attempts to perform track reconstruction. In general, this is simply used as a step to then perform the Time Based reconstruction.

### B.2.1 Hits to Clusters

Raw hits are all initially sorted into clumps. Clumps are the rawest form of a cluster as they are created by independently taking each superlayer in each sector and adding the hits found therein to the clump. The clump is then processed to remove hits caused by noise. This is determined by whether or not the DOCA for that hit is greater than the size of the cell. The list of clumps is then processed into a list of clusters by removing any clumps which have less than four total hits within them. A linear fit is then performed on the hits which comprise each cluster. The clusters are split into multiple clusters if necessary by means of checking the  $\chi^2$  of a linear fit. Following that step, the program updates the raw hits with various additional information and stores them as well as the clusters.

### B.2.2 Clusters to Segments

Segments are built by first excluding any clusters with more than fourteen hits from consideration. Then, for each remaining cluster, the information from the linear fit is taken



from the tests done in the previous step and set in the segment accordingly. Some segments may then be culled if their average DOCA between the raw hit and the cluster fit line is too large.

### **B.2.3 Segments to Crosses**

Crosses are created by looping over all sectors and regions in order and identifying segments in the first superlayer of each region. For each of those segments, it then loops through again to find a corresponding segment in the second superlayer. A check is done to see if the segments are both physically close and have a consistent slope from one to the other. If the segments pass both of those requirements, the cross is created.

Crosses are then arranged into short lists which correspond to possible tracks. To do this, all of the crosses are first sorted into a list containing only crosses from its region. Then, for every possible combination of three crosses such that there is always a cross from each region, the algorithm performs an approximate fit. This fit returns the trajectory of the fit at each of the cross positions. If the angle between any pair of corresponding crosses and trajectories is too great, the triplet of crosses is not considered for a track reconstruction. Those that pass this test are then checked again. This time a projection of the track onto the x-y plane is checked for linearity. Any triplets remaining are now bases for track candidates.

### **B.2.4 Crosses to Tracks Part I**

The algorithm now takes the triplets of crosses from the previous step and attempts to fit actual tracks from them. For each triplet, the first step is to perform another fit, this time parametric. This fit is formed by constraining the parametric trajectory to be tangent to the momentum unit vector of each cross as it passes through them. Any triplets which fail to produce a satisfactory trajectory are culled. At this point the algorithm splits finding track candidates into two branches. The first is the case where the cross triplet is indeed a full triplet without any members missing, and the torus magnet has been scaled down to a value

near zero. This is looking for straight tracks in the torus region. Otherwise the algorithm will search for linked helices, one from the torus region and one from the solenoid.

## **Straight Tracks**

This sub-algorithm first fits the projections of the crosses onto both the x-z and y-z planes to lines. If the fit is ok for each of those, the track is then assumed to be straight and the interaction vertex is assumed to be wherever the linear fits intersect the  $z = 0$  plane. The particle is also assumed to be a muon. This track candidate is then further refined by passing it through a Kalman fitter. The candidate is then added to the list of track candidates.

## **Linked Helices**

First, track candidates that don't have at least five segments within their crosses are culled. Then four initial fits are performed, one for each combination of one of the region one segments being coupled with one of the region three segments. In these fits, the slopes of the segments are used to calculate the momentum and charge of the particle. Those values are then input into a swimmer which will swim from the region one cross to the x-y plane of the region two cross and then on to the x-y plane of the region three cross. If the  $\chi^2$  is bad or the integrated magnetic field over the path is zero, the candidate is culled.

The momentum and charge are reconstructed, again, from the second segment of the region 1 and 3 crosses. If the momentum is greater than 11GeV, it is set to 11GeV. A state vector is formed from combining the region 1 cross information with the momentum. A Kalman fitter is created and attempts to fit the crosses. If this fails, the candidate is culled. The  $\chi^2$  of the fit between the Kalman fit and the crosses is calculated, and the candidate is culled if the value is too large. One final check to see if the Kalman fit either failed or if the final state vector produced by it is done. If passed, the candidate is added to the list of track candidates.

### B.2.5 Crosses to Tracks Part II

Now that the sub-algorithms have finished, the overall algorithm now attempts to extrapolate from the existing information to see if any other track candidates can be constructed. First, any tracks overlapping the track with the best  $\chi^2$  are culled. The algorithm then constructs roads out of all of the currently existing segments. This process, in essence, simply orders the segments in terms of sectors and superlayers then finds sequences of them which could indicate a track. A list of crosses belonging to those roads is then constructed. Missing road segments, called pseudosegments, are extrapolated from the existing segments.

The pseudosegments are added to the preexisting list of segments, and the process of making crosses is undertaken again. These new crosses are combined with the old ones and used to create new cross triplets. These triplets are then fed into the track candidate finder, as per Crosses to Tracks Part I. Once again, overlapping tracks are culled. If any track candidates have survived, they are written to the Hit Based data banks.

## B.3 Time Based Algorithm

The Time Based algorithm takes the output of the Hit Based Algorithm and includes more time information in order to refine the reconstruction.

### B.3.1 Hits to Clusters

This reconstruction begins by recomposing the clusters from the Hit Based Reconstruction and further refining them. First, for each cluster the algorithm removes secondary hits, which are defined as hits that lie within the same layer as another hit and fail a test comparing the relative DOCA's. The clusters are then refit with the aforementioned  $\chi^2$  test if necessary to ensure that the most accurate clusters have been found. Next, the algorithm attempts to resolve the left-right ambiguity in the cluster's linear fit, i.e. determine on which side of which wires the particle passed. Finally, the clusters and associated hits are updated with this new information.

### **B.3.2 Clusters to Segments**

Each cluster is subjected to a test wherein the average DOCA of the hits within the cluster is calculated and then compared to the average cell size of the cells where those hits were recorded. Clusters which fail this test are culled. The clusters then undergo the same process as the Hit Based clusters. All of the Hit Based tracks are now read into memory and the hit-segment associations are set accordingly.

### **B.3.3 Reconstructing Tracks**

The algorithm now begins track reconstruction, failing if the Hit Based Tracks were not saved correctly. First, all of the Hit Based track information stored in the data banks are read into an array, and all of the segment-track associations are set. The algorithm now considers each track, failing if the track has less than four segments, and creating crosses in the same fashion as the Hit Based algorithm. These tracks are now run through a Kalman fitter. The track is culled if the interaction vertex is too large. Finally, for each surviving track, the overall trajectory is calculated and associated hits and segments are set accordingly.

# Appendix C

## Source Code For Raster Based Reconstruction

This appendix contains the source code for the Raster Based reconstruction algorithm. There are some minor cosmetic changes to improve readability in this document.

```
1 package org.jlab.service.dc;
2
3 import java.io.File;
4 import java.io.IOException;
5 import java.util.ArrayList;
6 import java.util.Arrays;
7 import java.util.concurrent.Callable;
8 import java.util.concurrent.ExecutionException;
9 import java.util.concurrent.ExecutorService;
10 import java.util.concurrent.Executors;
11 import java.util.concurrent.Future;
12 import java.util.concurrent.TimeUnit;
13 import java.util.logging.Level;
14 import java.util.logging.Logger;
15 import org.jlab.clas.swimtools.MagFieldsEngine;
16 import org.jlab.clas.swimtools.Swim;
17 import org.jlab.clas.swimtools.Swimmer;
```

```

18 import org.jlab.io.base.DataBank;
19 import org.jlab.io.base.DataEvent;
20 import org.jlab.io.hipo.HipoDataEvent;
21 import org.jlab.io.hipo.HipoDataSource;
22 import org.jlab.io.hipo.HipoDataSync;
23 import org.jlab.jnp.hipo.schema.SchemaFactory;
24 import org.jlab.rec.dc.Constants;
25
26 /**
27  * This class is intended to be used to reconstruct the interaction vertex
28  *   ↪ from
29  * previously calculated HB/TB and CVT data
30  *
31  * @author friant
32  */
33 public class DCRBEngine extends DCEngine {
34     //Global Variables
35     static String engine          = "";
36     static float  solVal          = -1.0f * -1.0f; //-1.0 multiplier to
37     //   ↪ correct for sign convention mismatch
38     static float  torVal          = -1.0f;
39     static float  zMinGlobal      = -10.0f;
40     static float  zMaxGlobal      = 10.0f;
41     static float  percentGridSearch = 0.10f;
42     static int    samplesGridSearch = 5;
43     static int    iterationsVertex = 2;
44     static int    samplesVertex    = 5;
45     static int    maxThreads       = 1;
46
47     /**
48     * Constructor
49     */
50     public DCRBEngine() {
51         super("DCRB");
52     }
53 }

```

```

50     }
51
52     /**
53      * Main method
54      *
55      * @param args
56      */
57     public static void main(String[] args){
58         //Ensure that RasterBased Data Banks are defined in the program
59         SchemaFactory fact = new SchemaFactory();
60         fact.initFromDirectory("CLAS12DIR", "etc/bankdefs/hipo");
61
62         //Hipo Reader and Writer
63         HipoDataSource reader = new HipoDataSource();
64         HipoDataSync writer = new HipoDataSync(fact);
65
66         //Files
67         File inputFile;
68         File outputFile;
69
70         //Command Line Options
71         boolean help = false;
72
73         //Required
74         String input = "";
75         String output = "";
76
77         //Optional
78         String gridSearchSamples = "";
79         String gridSearchPercent = "";
80         String solenoid = "";
81         String toroid = "";
82         String vertexSamples = "";
83         String vertexIterations = "";

```

```

84     String threads = "";
85     String lowerBound = "";
86     String upperBound = "";
87
88     //Parse
89     for(int i = 0; i < args.length; i++){
90         switch(args[i]){
91             case "-i":
92                 if(i + 1 < args.length){
93                     input = args[i + 1];
94                 }
95                 else{
96                     System.out.println("Missing Parameter For Option -i");
97                     return;
98                 }
99                 break;
100            case "-o":
101                if(i + 1 < args.length){
102                    output = args[i + 1];
103                }
104                else{
105                    System.out.println("Missing Parameter For Option -o");
106                    return;
107                }
108                break;
109            case "-e":
110                if(i + 1 < args.length){
111                    engine = args[i + 1];
112                }
113                else{
114                    System.out.println("Missing Parameter For Option -e");
115                    return;
116                }
117                break;

```



```

118         case "-g":
119             if(i + 2 < args.length){
120                 gridSearchSamples = args[i + 1];
121                 gridSearchPercent = args[i + 2];
122             }
123             else{
124                 System.out.println("Missing Parameter For Option -g");
125                 return;
126             }
127             break;
128         case "-h":
129             help = true;
130             break;
131         case "-s":
132             if(i + 1 < args.length){
133                 solenoid = args[i + 1];
134             }
135             else{
136                 System.out.println("Missing Parameter For Option -s");
137                 return;
138             }
139             break;
140         case "-t":
141             if(i + 1 < args.length){
142                 toroid = args[i + 1];
143             }
144             else{
145                 System.out.println("Missing Parameter For Option -t");
146                 return;
147             }
148             break;
149         case "-v":
150             if(i + 2 < args.length){
151                 vertexSamples = args[i + 1];

```

```

152         vertexIterations = args[i + 2];
153     }
154     else{
155         System.out.println("Missing Parameter For Option -v");
156         return;
157     }
158     break;
159     case "-x":
160         if(i + 1 < args.length){
161             threads = args[i + 1];
162         }
163         else{
164             System.out.println("Missing Parameter For Option -x");
165             return;
166         }
167         break;
168     case "-z":
169         if(i + 2 < args.length){
170             lowerBound = args[i + 1];
171             upperBound = args[i + 2];
172         }
173         else{
174             System.out.println("Missing Parameter For Option -z");
175             return;
176         }
177         break;
178     }
179 }
180
181 //Attempt to use command line parameters to set values
182 //Input File
183 if(input.isEmpty()){
184     System.out.println("Input File Required");
185     help = true;

```

```

186     }
187     else{
188         inputFile = new File(input);
189         if(inputFile.exists() && !inputFile.isDirectory()){
190             reader.open(inputFile);
191         }
192         else{
193             System.out.println("Input File Not Found");
194             return;
195         }
196     }
197     ////output File
198     if(output.isEmpty()){
199         System.out.println("Output File Required");
200         help = true;
201     }
202     else{
203         outputFile = new File(output);
204         if(outputFile.exists() && !outputFile.isDirectory()){
205             outputFile.delete();
206         }
207         try{
208             outputFile.createNewFile();
209             writer.open(outputFile.getAbsolutePath());
210         }
211         catch(IOException e){
212             System.out.println("Could Not Create Output File");
213             return;
214         }
215     }
216     ////Engine
217     if(!engine.isEmpty()){
218         if(!(engine.equals("DCHB") || engine.equals("DCTB"))){
219             System.out.println("Invalid Engine Specified");

```

```

220         help = true;
221     }
222 }
223 ////Grid Search Parameters
224 if(!(gridSearchSamples.isEmpty() || gridSearchPercent.isEmpty())){
225     try{
226         samplesGridSearch = Integer.parseInt(gridSearchSamples);
227         percentGridSearch = Float.parseFloat(gridSearchPercent);
228     }
229     catch(NumberFormatException e){
230         System.out.println("Invalid Number Format For Grid Search
231                               ↪ Parameters");
232         help = true;
233     }
234 }
235 ////Solenoid Scale
236 if(!solenoid.isEmpty()){
237     try{
238         solVal = Float.parseFloat(solenoid);
239         solVal = -1.0f * solVal;//-1.0 multiplier to correct for sign
240         ↪ convention mismatch
241     }
242     catch(NumberFormatException e){
243         System.out.println("Invalid Number Format For Solenoid");
244         help = true;
245     }
246 }
247 ////Toroid Scale
248 if(!toroid.isEmpty()){
249     try{
250         torVal = Float.parseFloat(toroid);
251     }
252     catch(NumberFormatException e){
253         System.out.println("Invalid Number Format For Toroid");

```

```

252         help = true;
253     }
254 }
255 ////Vertex Search Parameters
256 if(!(vertexSamples.isEmpty() || vertexIterations.isEmpty())){
257     try{
258         samplesVertex = Integer.parseInt(vertexSamples);
259         iterationsVertex = Integer.parseInt(vertexIterations);
260     }
261     catch(NumberFormatException e){
262         System.out.println("Invalid Number Format For Vertex Search
                ↵ Parameters");
263         help = true;
264     }
265 }
266 ////Threads
267 if(!threads.isEmpty()){
268     try{
269         maxThreads = Integer.parseInt(threads);
270     }
271     catch(NumberFormatException e){
272         System.out.println("Invalid Number Format For Number Of
                ↵ Threads To Use");
273         help = true;
274     }
275 }
276 ////Target Bounds
277 if(!(lowerBound.isEmpty() || upperBound.isEmpty())){
278     try{
279         zMinGlobal = Float.parseFloat(lowerBound);
280         zMaxGlobal = Float.parseFloat(upperBound);
281     }
282     catch(NumberFormatException e){
283         System.out.println("Invalid Number Format For Target Bounds");

```

```

284         help = true;
285     }
286 }
287
288 //Print help message and exit
289 if(help){
290     printHelp();
291     return;
292 }
293
294 //Init Engines
295 MagFieldsEngine magField = new MagFieldsEngine();
296 magField.init();
297 DCRBEngine thisEngine = new DCRBEngine();
298 thisEngine.init();
299
300 //Apply magnetic field scaling
301 Swimmer.setMagneticFieldsScales(solVal, torVal, 0.0);
302
303 //Process data events
304 int count = 0;
305 while(reader.hasEvent()){
306     DataEvent event = reader.getNextEvent();
307     if(!event.hasBank("RasterBasedTrkg::RBHits")){
308         ((HipoDataEvent)event).initDictionary(fact);
309     }
310     thisEngine.processDataEvent(event);
311     writer.writeEvent(event);
312     System.out.println("EVENT " + count + " PROCESSED\r\n");
313     count++;
314 }
315
316 //Tidy up before leaving
317 reader.close();

```

```

318         writer.close();
319     }
320
321     /**
322      * Print help message to the terminal
323      */
324     private static void printHelp() {
325         System.out.println(
326             "FriTracking Command Line Options:                \r\n"
327             + " -Required:                \r\n"
328             + "     -i      Input Hipo File                \r\n"
329             + "           Requires 1 Parameter:            \r\n"
330             + "               1 (String): Path to desired input file.    \r\n"
331             + "           Ex: -i /path/to/my/input/file.hipo            \r\n"
332             + "           \r\n"
333             + "     -o      Output Hipo File                \r\n"
334             + "           Requires 1 Parameter:            \r\n"
335             + "               1 (String): Path to the desired output file. \r\n"
336             + "           Will overwrite file if exists.    \r\n"
337             + "           Ex: -o /path/to/my/output/file.hipo            \r\n"
338             + "           \r\n"
339             + " -Optional:                \r\n"
340             + "     -e      Engine To Source Data From        \r\n"
341             + "           Requires 1 Parameter:            \r\n"
342             + "               1 (String): Either \"DCHB\" or \"DCTB\"        \r\n"
343             + "           Default Behavior:                \r\n"
344             + "               Use DCTB if available and DCHB if not.    \r\n"
345             + "           Ex: -e DCHB                \r\n"
346             + "           \r\n"
347             + "     -g      Grid Search Parameters            \r\n"
348             + "           Requires 2 Parameters:            \r\n"
349             + "               1 (Int ): The number of samples to take per\r\n"
350             + "               momentum dimension around the    \r\n"
351             + "               region-1 cross.                \r\n"

```

```

352      + "                2 (Float ): The percent of the nominal value \r\n"
353      + "                to search within. \r\n"
354      + "      Default Behavior: \r\n"
355      + "                5 Samples within 10% of the nominal value \r\n"
356      + "      Ex: -g 7 0.02 \r\n"
357      + " \r\n"
358      + "      -h      Print This Message \r\n"
359      + " \r\n"
360      + "      -s      Solenoid Field Scale Factor \r\n"
361      + "      Requires 1 Parameter \r\n"
362      + "                1 (Float ): The number by which to scale the \r\n"
363      + "                solenoid's magnetic field. Note: \r\n"
364      + "                a value of 0.0 will likely cause \r\n"
365      + "                failure. \r\n"
366      + "      Default Behavior: \r\n"
367      + "                Scale by -1.0. \r\n"
368      + "      Ex: -s 0.05 \r\n"
369      + " \r\n"
370      + "      -t      Toroid Field Scale Factor \r\n"
371      + "      Requires 1 Parameter \r\n"
372      + "                1 (Float ): The number by which to scale the \r\n"
373      + "                toroid's magnetic field. Note: a \r\n"
374      + "                value of 0.0 will likely cause \r\n"
375      + "                failure. \r\n"
376      + "      Default Behavior: \r\n"
377      + "                Scale by -1.0. \r\n"
378      + "      Ex: -s 0.1 \r\n"
379      + " \r\n"
380      + "      -v      Vertex Search Parameters \r\n"
381      + "      Requires 2 Parameters: \r\n"
382      + "                1 (Int ): The number of samples to take \r\n"
383      + "                within the target range to find \r\n"
384      + "                the best z value. \r\n"
385      + "                2 (Int ): The number of recursive \r\n"

```



```

386         + "                iterations to go through about \r\n"
387         + "                each local minima. \r\n"
388         + "        Default Behavior: \r\n"
389         + "                5 samples with 2 iterations. \r\n"
390         + "        Ex: -v 10 2 \r\n"
391         + " \r\n"
392         + "        -x    Threads To Use \r\n"
393         + "        Requires 1 Parameter: \r\n"
394         + "                1 (Int ): The maximum number of threads to \r\n"
395         + "                use while searching for minima. \r\n"
396         + "        Default Behavior: \r\n"
397         + "                Only use 1 thread. \r\n"
398         + "        Ex: -x 4 \r\n"
399         + " \r\n"
400         + "        -z    Target Bounds \r\n"
401         + "        Requires 2 Parameters: \r\n"
402         + "                1 (Float ): The lower bound in the \r\n"
403         + "                z-direction in which to search \r\n"
404         + "                for the interaction vertex. \r\n"
405         + "                [Unit = cm] \r\n"
406         + "                2 (Float ): The upper bound in the \r\n"
407         + "                z-direction in which to search \r\n"
408         + "                for the interaction vertex. \r\n"
409         + "                [Unit = cm] \r\n"
410         + "        Default Behavior: \r\n"
411         + "                Lower Bound = -10.0, Upper Bound = 10.0. \r\n"
412         + "        Ex: -z -2.5 5.0 \r\n"
413         + "\r\n");
414     }
415
416     /**
417     * Initialize the engine
418     *
419     * @return

```

```

420     */
421     @Override
422     public boolean init() {
423         // Load cuts
424         Constants.Load();
425         super.setStartTimeOption();
426         super.LoadTables();
427         return true;
428     }
429
430     /**
431      * Generic process data event function
432      *
433      * @param event
434      * @return
435      */
436     @Override
437     public boolean processDataEvent(DataEvent event){
438         boolean hasCVTData = event.hasBank("CVTRec::Tracks");
439         boolean hasHBData = event.hasBank("HitBasedTrkg::HBTracks");
440
441         if(hasCVTData && hasHBData){
442             //processDataEventBoth(event); //Just use HB for now
443             processDataEventHB(event);
444         }
445         else if(hasCVTData){
446             processDataEventCVT(event);
447         }
448         else if(hasHBData){
449             processDataEventHB(event);
450         }
451
452         return true;
453     }

```

```

454
455     /**
456      * Process data event for when HB and CVT data is available
457      *
458      * @param event
459      * @return
460      */
461     public boolean processDataEventBoth(DataEvent event){
462         System.out.println("Process for Both CVT and HB Not Yet Implemented");
463         return true;
464     }
465
466     /**
467      * Process data event for when only CVT data is available
468      *
469      * @param event
470      * @return
471      */
472     public boolean processDataEventCVT(DataEvent event){
473         System.out.println("Process for CVT Not Yet Implemented");
474         return true;
475     }
476
477     /**
478      * Process data event for when only HB/TB data is available
479      *
480      * @param event
481      * @return
482      */
483     public boolean processDataEventHB(DataEvent event) {
484         //Pull info out of TB/HB Banks
485         String sourceTracks;
486         String sourceCrosses;
487         if(engine.isEmpty()){

```

```

488     if (event.hasBank("TimeBasedTrkg::TBTracks")) {
489         event.appendBank(copyBank(event, "TimeBasedTrkg::TBHits",
490             ↪ "RasterBasedTrkg::RBHits"));
491         event.appendBank(copyBank(event, "TimeBasedTrkg::TBClusters",
492             ↪ "RasterBasedTrkg::RBClusters"));
493         event.appendBank(copyBank(event, "TimeBasedTrkg::TBSegments",
494             ↪ "RasterBasedTrkg::RBSegments"));
495         event.appendBank(copyBank(event, "TimeBasedTrkg::TBCrosses",
496             ↪ "RasterBasedTrkg::RBCrosses"));
497         sourceTracks = "TimeBasedTrkg::TBTracks";
498         sourceCrosses = "TimeBasedTrkg::TBCrosses";
499     }
500     else {
501         event.appendBank(copyBank(event, "HitBasedTrkg::HBHits",      "
502             ↪ RasterBasedTrkg::RBHits"));
503         event.appendBank(copyBank(event, "HitBasedTrkg::HBClusters",
504             ↪ "RasterBasedTrkg::RBClusters"));
505         event.appendBank(copyBank(event, "HitBasedTrkg::HBSegments",
506             ↪ "RasterBasedTrkg::RBSegments"));
507         event.appendBank(copyBank(event, "HitBasedTrkg::HBCrosses",
508             ↪ "RasterBasedTrkg::RBCrosses"));
509         sourceTracks = "HitBasedTrkg::HBTracks";
510         sourceCrosses = "HitBasedTrkg::HBCrosses";
511     }
512 }
513
514 else if (engine.equals("DCHB")) {
515     if (event.hasBank("TimeBasedTrkg::TBTracks")) {
516         event.appendBank(copyBank(event, "HitBasedTrkg::HBHits",
517             ↪ "RasterBasedTrkg::RBHits"));
518         event.appendBank(copyBank(event, "HitBasedTrkg::HBClusters",
519             ↪ "RasterBasedTrkg::RBClusters"));
520         event.appendBank(copyBank(event, "HitBasedTrkg::HBSegments",
521             ↪ "RasterBasedTrkg::RBSegments"));

```

```

510         event.appendBank(copyBank(event, "HitBasedTrkg::HBCrosses",
511                                   ⇨ "RasterBasedTrkg::RBCrosses"));
512         sourceTracks = "HitBasedTrkg::HBTracks";
513         sourceCrosses = "HitBasedTrkg::HBCrosses";
514     }
515     else{
516         return false;
517     }
518     else if(engine.equals("DCTB")){
519         if(event.hasBank("TimeBasedTrkg::TBTracks")){
520             event.appendBank(copyBank(event, "TimeBasedTrkg::TBHits",
521                                       ⇨ "RasterBasedTrkg::RBHits"));
522             event.appendBank(copyBank(event, "TimeBasedTrkg::TBClusters",
523                                       ⇨ "RasterBasedTrkg::RBClusters"));
524             event.appendBank(copyBank(event, "TimeBasedTrkg::TBSegments",
525                                       ⇨ "RasterBasedTrkg::RBSegments"));
526             event.appendBank(copyBank(event, "TimeBasedTrkg::TBCrosses",
527                                       ⇨ "RasterBasedTrkg::RBCrosses"));
528             sourceTracks = "TimeBasedTrkg::TBTracks";
529             sourceCrosses = "TimeBasedTrkg::TBCrosses";
530         }
531         else{
532             return false;
533         }
534     }
535     else{
536         return false;
537     }
538
539     //Create the RBTracks Bank
540     DataBank rbBank = copyBank(event, sourceTracks,
541                                "RasterBasedTrkg::RBTracks");

```

```

539      //Raster variables
540      float rasterX = 0.0f;
541      float rasterY = 0.0f;
542      float rasterUX = 0.05f;//0.5mm uncertainty in either direction
543      float rasterUY = 0.05f;//0.5mm uncertainty in either direction
544      if(event.hasBank("MC::Particle")){
545          rasterX = event.getBank("MC::Particle").getFloat("vx", 0);
546          rasterY = event.getBank("MC::Particle").getFloat("vy", 0);
547      }
548      float[] rasterInfo = new float[]{rasterX, rasterUX,
549                                         rasterY, rasterUY};
550
551      //Calculate the interaction vertex for each for each track in the
552      ↪ event
553      for(int i = 0; i < rbBank.rows(); i++){
554          float[] trackInfo = new float[7];
555          trackInfo[0] = event.getBank(sourceTracks).getFloat("t1_x" , i);
556          trackInfo[1] = event.getBank(sourceTracks).getFloat("t1_y" , i);
557          trackInfo[2] = event.getBank(sourceTracks).getFloat("t1_z" , i);
558          trackInfo[3] = event.getBank(sourceTracks).getFloat("t1_px", i);
559          trackInfo[4] = event.getBank(sourceTracks).getFloat("t1_py", i);
560          trackInfo[5] = event.getBank(sourceTracks).getFloat("t1_pz", i);
561          trackInfo[6] = (float) (event.getBank(sourceTracks).getBytes("q", i)
562                                  ↪ );
563
564      //Calculate
565      float[] output = new float[8];
566      Arrays.fill(output, Float.NaN);
567      float doca = -1.0f;
568      try {
569          doca = interactionVertexGridSearch(samplesGridSearch,
570                                              ↪ rasterInfo, trackInfo, percentGridSearch, output);
571      } catch (InterruptedException | ExecutionException exception) {

```

```

569         Logger.getLogger(DCRBEngine.class.getName()).log(Level.SEVERE,
           ↪ null, exception);
570         System.out.println(exception);
571     }
572
573     //Make sure that the momentum is pointing in the right direction
574     if(output[5] < 0.0){
575         output[3] = output[3] * -1.0f;
576         output[4] = output[4] * -1.0f;
577         output[5] = output[5] * -1.0f;
578     }
579
580     //System.out.println("Doca: " + doca);
581     //System.out.println(Arrays.toString(output));
582
583     //Set Values
584     rbBank.setFloat("Vtx0_x", i, output[0]);
585     rbBank.setFloat("Vtx0_y", i, output[1]);
586     rbBank.setFloat("Vtx0_z", i, output[2]);
587     rbBank.setFloat("p0_x", i, output[3]);
588     rbBank.setFloat("p0_y", i, output[4]);
589     rbBank.setFloat("p0_z", i, output[5]);
590     rbBank.setFloat("doca", i, doca);
591 }
592
593 //Put Tracks in Event
594 event.appendBank(rbBank);
595
596     return true;
597 }
598
599 /**
600     * This method performs a three dimensional grid search within the
601     * uncertainty in the track's momentum.

```

```

602      *
603      * @param samples      the number of samples to ake in each dimension
604      * @param rasterInfo   the array {x, uncert x, y, uncert y}
605      * @param trackInfo    the array{vx, vy, vz, px, py, pz}
606      * @param uncertainty  + or - percent uncertainty in track momentum
607      * @param out          the array to fill with the ultimate swim output
608      *                    result
609      * @return             the doca of the ultimate swim output result
610      */
611      private float interactionVertexGridSearch(int samples, float[] rasterInfo,
        ↪ float[] trackInfo, float uncertainty, float[] out) throws
        ↪ InterruptedException, ExecutionException{
612          //define bounds
613          float upperBoundPX = trackInfo[3] * (1.0f + uncertainty);
614          float lowerBoundPX = trackInfo[3] * (1.0f - uncertainty);
615          float upperBoundPY = trackInfo[4] * (1.0f + uncertainty);
616          float lowerBoundPY = trackInfo[4] * (1.0f - uncertainty);
617          float upperBoundPZ = trackInfo[5] * (1.0f + uncertainty);
618          float lowerBoundPZ = trackInfo[5] * (1.0f - uncertainty);
619
620          //choose the max radius of uncertainty to check doca against
621          float rasterErr = Math.max(rasterInfo[1], rasterInfo[3]);
622
623          //define useful arrays
624          float[][][][] output = new float[samples][samples][samples][8];
625          ArrayList<ArrayList<ArrayList<Future<Float>>>> doca = new ArrayList();
626          ArrayList<int[]> localMinIndices = new ArrayList<>();
627
628          //Define thread service
629          ExecutorService ex = Executors.newFixedThreadPool(maxThreads);
630
631          //Find the interaction vertices and docas to sift through
632          for(int i = 0; i < samples; i++){
633              doca.add(new ArrayList<>());

```



```

634     for(int j = 0; j < samples; j++){
635         doca.get(i).add(new ArrayList<>());
636         for(int k = 0; k < samples; k++){
637             //Set Swimmer params
638             float[] swimParams = new float[]{
639                 trackInfo[0],
640                 trackInfo[1],
641                 trackInfo[2],
642                 lowerBoundPX + i * (upperBoundPX - lowerBoundPX) /
643                     ↪ (samples - 1),
644                 lowerBoundPY + j * (upperBoundPY - lowerBoundPY) /
645                     ↪ (samples - 1),
646                 lowerBoundPZ + k * (upperBoundPZ - lowerBoundPZ) /
647                     ↪ (samples - 1),
648                 trackInfo[6]};
649
650             //Get interaction vertices
651             doca.get(i).get(j).add(ex.submit(new ThreadedVertexFinder(
652                 iterationsVertex,
653                 samplesVertex,
654                 zMinGlobal,
655                 zMaxGlobal,
656                 rasterInfo[0],
657                 rasterInfo[2],
658                 swimParams,
659                 output[i][j][k]))));
660         }
661     }
662 }
663
664 //Wait for all tasks to finish
665 ex.shutdown();
666 ex.awaitTermination(10, TimeUnit.DAYS);//Arbitrarily large

```

```

665      //Find local mins
666      for(int i = 0; i < samples; i++){
667          for(int j = 0; j < samples; j++){
668              for(int k = 0; k < samples; k++){
669                  //check if within rasterErr
670                  if(doca.get(i).get(j).get(k).get() < rasterErr){
671                      localMinIndices.add(new int[]{i, j, k});
672                      continue;
673                  }
674                  //check if unrealistically massive
675                  if(doca.get(i).get(j).get(k).get() > Float.MAX_VALUE /
676                      ↪ 2.0){
677                      continue;
678                  }
679                  //check px component
680                  if(i == 0 && doca.get(i).get(j).get(k).get() > doca.get(i
681                      ↪ + 1).get(j).get(k).get())){
682                      continue;
683                  }
684                  else if (i > 0 && i < samples - 1 && (doca.get(i).get(j).
685                      ↪ get(k).get() > doca.get(i + 1).get(j).get(k).get()
686                      ↪ || doca.get(i).get(j).get(k).get() > doca.get(i - 1)
687                      ↪ .get(j).get(k).get()))){
688                      continue;
689                  }
690                  //check py component
691                  if(j == 0 && doca.get(i).get(j).get(k).get() > doca.get(i)
692                      ↪ .get(j + 1).get(k).get())){
693                      continue;
694                  }

```

```

692         else if (j > 0 && j < samples - 1 && (doca.get(i).get(j).
        ↪ get(k).get() > doca.get(i).get(j + 1).get(k).get()
        ↪ || doca.get(i).get(j).get(k).get() > doca.get(i).get
        ↪ (j - 1).get(k).get())){
693             continue;
694         }
695         else if(j == samples - 1 && doca.get(i).get(j).get(k).get
        ↪ () > doca.get(i).get(j - 1).get(k).get()){
696             continue;
697         }
698         //check pz component
699         if(k == 0 && doca.get(i).get(j).get(k).get() > doca.get(i)
        ↪ .get(j).get(k + 1).get()){
700             continue;
701         }
702         else if (k > 0 && k < samples - 1 && (doca.get(i).get(j).
        ↪ get(k).get() > doca.get(i).get(j).get(k + 1).get()
        ↪ || doca.get(i).get(j).get(k).get() > doca.get(i).get
        ↪ (j).get(k - 1).get())){
703             continue;
704         }
705         else if(k == samples - 1 && doca.get(i).get(j).get(k).get
        ↪ () > doca.get(i).get(j).get(k - 1).get()){
706             continue;
707         }
708         //add to local min list
709         localMinIndices.add(new int[]{i, j, k});
710     }
711 }
712 }
713
714 //Cull values that fail zMinGlobal < z < zMaxGlobal
715 for(int i = 0; i < localMinIndices.size(); i++){
716     int[] idx = localMinIndices.get(i);

```

```

717         if(output[idx[0]][idx[1]][idx[2]][2] < zMinGlobal || output[idx
           ↪ [0]][idx[1]][idx[2]][2] > zMaxGlobal){
718             localMinIndices.remove(i);
719             i--;
720         }
721     }
722
723     //Exit
724     int[] i;
725     switch(localMinIndices.size()){
726         case 0:
727             //Fail gracefully
728             Arrays.fill(out, Float.NaN);
729             return Float.NaN;
730         case 1:
731             //Only one value
732             i = localMinIndices.get(0);
733             System.arraycopy(output[i[0]][i[1]][i[2]], 0, out, 0, 8);
734             return doca.get(i[0]).get(i[1]).get(i[2]).get();
735         default:
736             //Find value which minimizes change from nominal track
737             float minIdxDiff = Float.MAX_VALUE;
738             float minDoca = Float.MAX_VALUE;
739             boolean inRasterErr = false;
740             int minIndex = 0;
741             for(int[] idx : localMinIndices){
742                 float swimDoca = doca.get(idx[0]).get(idx[1]).get(idx[2]).
           ↪ get();
743                 //System.out.println("idx: " + Arrays.toString(idx) + ",
           ↪ doca: " + swimDoca + ", z: " + output[idx[0]][idx
           ↪ [1]][idx[2]][2]);
744                 if(swimDoca < rasterErr){
745                     inRasterErr = true;

```

```

746         float idxDiff = (float)Math.sqrt(Math.pow(idx[0] - (
           ↪ float)samples / 2, 2.0) +
747                                     Math.pow(idx[1] - (
           ↪ float)samples /
           ↪ 2, 2.0) +
748                                     Math.pow(idx[2] - (
           ↪ float)samples /
           ↪ 2, 2.0));

749         if(idxDiff < minIdxDiff){
750             minIdxDiff = idxDiff;
751             minIndex = localMinIndices.indexOf(idx);
752         }
753     }
754     else if(inRasterErr == false){
755         if(swimDoca < minDoca){
756             minDoca = swimDoca;
757             minIndex = localMinIndices.indexOf(idx);
758         }
759     }
760
761     }
762     i = localMinIndices.get(minIndex);
763     //System.out.println("Winner: " + Arrays.toString(i));
764     System.arraycopy(output[i[0]][i[1]][i[2]], 0, out, 0, 8);
765     return doca.get(i[0]).get(i[1]).get(i[2]).get();
766 }
767 }
768
769 /**
770  * This method uses a swimmer to calculate the beam's doca position
       ↪ relative to the raster beam axis.
771  *
772  * @param iterations    The maximum number of recursive steps

```

```

773      * @param samples      The number of sample points to take between zMin
        ↪ and zMax for each step.
774      * @param zMin          The lower bound of the area of interest. Should be
        ↪ below the lower bound of the target
775      * @param zMax          The upper bound of the area of interest. Should be
        ↪ above the upper bound of the target
776      * @param rasterX       The rasterized beam x position
777      * @param rasterY       The rasterized beam y position
778      * @param swim          A Swim class which has been set to the position
        ↪ and momentum of interest
779      * @param out           A pointer to a float array which will be filled by
        ↪ this method. Will be set to the Swim output or NaN
780      * @return the doca to the rasterized beam coords; -1.0 if no minimum was
        ↪ found
781      */
782      private float findInteractionVertex(int iterations, int samples, float
        ↪ zMin, float zMax, float rasterX, float rasterY, Swim swim, float[]
        ↪ out){
783          //Define useful arrays
784          float[][] swimOutput = new float[samples][8];
785          float[] doca = new float[samples];
786          int[] docaIndex = new int[samples];
787          int docaLength = 0;
788          int[] localMin = new int[samples];
789          int localMinLength = 0;
790
791          //Get swim outputs
792          for(int i = 0; i < samples; i++){
793              double[] temp = swim.SwimToPlaneLab(zMin + i * (zMax - zMin) / (
        ↪ samples - 1));
794              if(temp == null){
795                  swimOutput[i] = null;
796                  continue;
797              }

```

```

798         for(int j = 0; j < 8; j++){
799             swimOutput[i][j] = (float)temp[j];
800         }
801     }
802
803     //Calculate the doca for each sample point
804     for(int i = 0; i < samples; i++){
805         if(swimOutput[i] == null){
806             continue;
807         }
808         doca[docaLength] = (float)Math.sqrt(Math.pow(rasterX - swimOutput[
            ↪ i][0], 2.0f) + Math.pow(rasterY - swimOutput[i][1], 2.0f));
809         docaIndex[docaLength] = i;
810         docaLength++;
811     }
812
813     //Handle variable docaLengths
814     switch(docaLength){
815         case 0:
816             //Fail gracefully
817             localMinLength = 0;
818             break;
819         case 1:
820             //Set only point as local minimum
821             localMinLength = 1;
822             localMin[0] = 0;
823             break;
824         default:
825             //Find local minima
826             if(doca[0] < doca[1]){
827                 localMin[localMinLength] = 0;
828                 localMinLength++;
829             } for(int i = 1; i < docaLength - 1; i++){
830                 if(doca[i] < doca[i - 1] && doca[i] < doca[i + 1]){

```

```

831             localMin[localMinLength] = i;
832             localMinLength++;
833         }
834     } if(doca[docaLength - 1] < doca[docaLength - 2]){
835         localMin[localMinLength] = docaLength - 1;
836         localMinLength++;
837     } break;
838 }
839
840 //Exit?
841 if(iterations == 1){
842     int index = -1;
843     float smallest = Float.MAX_VALUE;
844
845     //Find a minimum?
846     if(localMinLength == 0){
847         Arrays.fill(out, Float.NaN);
848         return Float.MAX_VALUE;
849     }
850
851     //Find the smallest doca
852     for(int i = 0; i < localMinLength; i++){
853         if(doca[localMin[i]] < smallest){
854             index = i;
855         }
856     }
857     System.arraycopy(swimOutput[localMin[index]], 0, out, 0, 8);
858     return doca[localMin[index]];
859 }
860
861 //Recursively call this method on each of the local minima
862 float[][] minOut = new float[localMinLength][8];
863 float[] minDoca = new float[localMinLength];
864 for(int i = 0; i < localMinLength; i++){

```



```

865     if(docaIndex[localMin[i]] == 0){
866         float newZMin = zMin - (zMax - zMin) / 2;
867         float newZMax = zMax - (zMax - zMin) / 2;
868         if(newZMin < 2.0 * zMinGlobal - zMaxGlobal){
869             Arrays.fill(minOut[i], Float.NaN);
870             minDoca[i] = Float.MAX_VALUE;
871         }
872         else{
873             minDoca[i] = findInteractionVertex(iterations, samples,
874                 ↪ newZMin, newZMax, rasterX, rasterY, swim, minOut[i])
875                 ↪ ;
876         }
877     }
878     else if(docaIndex[localMin[i]] == samples - 1){
879         float newZMin = zMin + (zMax - zMin) / 2;
880         float newZMax = zMax + (zMax - zMin) / 2;
881         if(newZMin > 2.0 * zMaxGlobal - zMinGlobal){
882             Arrays.fill(minOut[i], Float.NaN);
883             minDoca[i] = Float.MAX_VALUE;
884         }
885         else{
886             minDoca[i] = findInteractionVertex(iterations, samples,
887                 ↪ newZMin, newZMax, rasterX, rasterY, swim, minOut[i])
888                 ↪ ;
889         }
890     }
891     else{
892         minDoca[i] = findInteractionVertex(iterations - 1, samples,
893             ↪ swimOutput[localMin[i]][2] - (zMax - zMin) / (samples -
894             ↪ 1), swimOutput[localMin[i]][2] + (zMax - zMin) / (
895             ↪ samples - 1), rasterX, rasterY, swim, minOut[i]);
896     }
897 }

```

```

892      //Find the smallest doca
893      int index = -1;
894      float smallest = Float.MAX_VALUE;
895      for(int i = 0; i < localMinLength; i++){
896          if(minOut[i] != null && minDoca[i] < smallest)
897          {
898              index = i;
899          }
900      }
901
902      //Exit
903      if(index == -1){
904          Arrays.fill(out, Float.NaN);
905          return Float.MAX_VALUE;
906      }
907      System.arraycopy(minOut[index], 0, out, 0, 8);
908      return minDoca[index];
909  }
910
911  /**
912   * This class is a passthrough class to allow the findInteractionVertex
913   * method to be threaded
914   */
915  private class ThreadedVertexFinder implements Callable<Float>{
916      //Copies of method parameters
917      int iterations;
918      int samples;
919      float zMin;
920      float zMax;
921      float rasterX;
922      float rasterY;
923      Swim swim;
924      float[] out;
925

```

```

926         //Constructor
927         public ThreadedVertexFinder(int _iterations, int _samples, float _zMin
           ↪ , float _zMax, float _rasterX, float _rasterY, float[]
           ↪ _swimParams, float[] _out){
928             iterations = _iterations;
929             samples = _samples;
930             zMin = _zMin;
931             zMax = _zMax;
932             rasterX = _rasterX;
933             rasterY = _rasterY;
934             swim = new Swim();
935             swim.SetSwimParameters(_swimParams[0], _swimParams[1], _swimParams
           ↪ [2], _swimParams[3], _swimParams[4], _swimParams[5], (int)
           ↪ _swimParams[6]);
936             out = _out;
937         }
938
939         //Passthrough
940         @Override
941         public Float call(){
942             return findInteractionVertex(iterations, samples, zMin, zMax,
           ↪ rasterX, rasterY, swim, out);
943         }
944     }
945
946     /**
947      * Provides a copy mechanism from a pre-existing dataBank to a new one.
948      * Note that this does not natively append the new bank to the old event.
949      *
950      * @param event
951      * @param oldBank
952      * @param newBank
953      */

```

```

954     private DataBank copyBank(DataEvent event, String oldBankName, String
        ↪ newBankName){
955         DataBank oldBank = event.getBank(oldBankName);
956         if(oldBank == null){
957             return null;
958         }
959
960         DataBank newBank = event.createBank(newBankName, oldBank.rows());
961         for(int i = 0; i < oldBank.rows(); i++){
962             for(int j = 0; j < oldBank.columns(); j++){
963                 switch(oldBank.getDescriptor().getProperty("type", oldBank.
                    ↪ getColumnList()[j])){
964                     case 1://Byte
965                         newBank.setByte(oldBank.getColumnList()[j], i, (byte)(
                            ↪ oldBank.getByte(oldBank.getColumnList()[j], i))
                            ↪ ;
966                         break;
967                     case 2://Short
968                         newBank.setShort(oldBank.getColumnList()[j], i, (short
                            ↪ )(oldBank.getShort(oldBank.getColumnList()[j], i
                            ↪ ));
969                         break;
970                     case 3://Int
971                         newBank.setInt(oldBank.getColumnList()[j], i, (int)(
                            ↪ oldBank.getInt(oldBank.getColumnList()[j], i)));
972                         break;
973                     case 4://Unused
974                         break;
975                     case 5://Float
976                         newBank.setFloat(oldBank.getColumnList()[j], i, (float
                            ↪ )(oldBank.getFloat(oldBank.getColumnList()[j], i
                            ↪ ));
977                         break;
978                     case 6://Double

```

```
979         newBank.setDouble(oldBank.getColumnList()[j], i, (
            ↪ double) (oldBank.getDouble(oldBank.getColumnList
            ↪ () [j], i)));
980         break;
981     }
982 }
983 }
984 return newBank;
985 }
986 }
```