Advances in Cosmic Ray Muon Tomography Reconstruction Algorithms

by

Richard Claude Hoch

A thesis
submitted to the College of Engineering at
Florida Institute of Technology
in partial fulfillment of the requirements
for the degree of

Master of Science
in
Computer Science

Melbourne, Florida
December 2009

Sign Page

ABSTRACT

TITLE: Advances in Cosmic Ray Muon Tomography Reconstruction Algorithms

AUTHOR: Richard Hoch

THESIS ADVISOR: Debasis Mitra, Ph.D.

Cosmic ray muons shower the earth at a rate of 1 per square centimeter per minute at sea level. Techniques have been created to use these highly penetrating particles as a means of non-intrusive inspection by using the multiple Coulomb scattering the particles experience as an information source. In this thesis the concept and theory of cosmic ray muon tomography are described. Past attempts at using muons for imaging are also discussed. Two reconstruction algorithms were implemented and tested on simulated data. These algorithms were based on past attempts at muon tomography, namely the point of closest approach (POCA) and an expectation maximization (EM) algorithm. Improvements were made to both algorithms to increase discrimination power and efficiency. The algorithms and implementation are presented in depth. The results they produced based on Monte Carlo simulations are also presented and analyzed.

Table of Contents

List of Figures viii
List of Tables x
Acknowledgements xi
Dedication xiii

Chapter 1 Introduction 1
1.1 Purpose of Study 1
1.1.1. What Is Muon Tomography? 1
1.1.2. Importance of Muon Tomography 2
1.2 Scope of Study 4
1.3 Thesis Outline 4

Chapter 2 Background Information 5
2.1 Introduction 5
2.2 Tomography 5
2.3 Muons 6
2.3.1. Basics 6
2.3.2. Cosmic Ray Muons 7
2.3.3. Muon Physics 10
2.3.4. Muon Detectors 13
2.4 Muon Tomography 16
2.4.1. Concept 16
2.4.2. Reconstruction Algorithms 16
2.5 Past Work 17
2.5.1. Muon Radiography 17
2.5.2. Muon Tomography 19

2.5.3. Emission Tomography 21

Chapter 3 Research Questions 23

3.1 Research Questions 23

3.1.1. What is the goal of this work? 23

3.1.2. What are the expected or wanted results? 23

3.1.3. What are the potential advantages of this approach over others? 24

Chapter 4 Reconstruction Algorithms 25

4.1 Overview 25

4.2 Point of Closest Approach 25

4.2.1. POCA Algorithm 27

4.2.2. POCA Analysis 28

4.3 Expectation Maximization 29

4.3.1. EM Model 30

4.3.2. EM Development 36

4.3.3. EM Algorithm 37

4.3.4. EM Analysis 37

4.3.5. Improvements 38

Chapter 5 Implementation and Methodology 42

5.1 Simulation Overview 42

5.2 Tools 42

5.2.1. Geant4 42

5.2.2. Cosmic-Ray Shower Generator (CRY) 44

5.2.3. ROOT 45

5.3 Muon Tomography Suite 46

5.3.1. Driver Implementation 46
5.3.2. POCA Implementation 48
5.3.3. EM Data Structure 49
5.3.4. EM Preprocessing Implementation 52
5.3.5. EM Implementation 53
5.4 Software Testing 54
5.4.1. General Techniques 55
5.4.2. Specific Examples of Software Testing 56

Chapter 6 Experiments and Results 60
6.1 Introduction 61
6.2 Scenarios 60
6.2.1. Basic Scenario 61
6.2.2. Five Target Scenario 62
6.2.3. LANL Scenario 63
6.2.4. Vertical Clutter Scenario 64
6.2.5. Truck Scenario 65
6.3 POCA Results 66
6.3.1. Basic Scenario 67
6.3.2. Five Target Scenario 68
6.3.3. LANL Scenario 69
6.3.4. Vertical Clutter Scenario 69
6.3.5. Truck Scenario 70
6.4 Average-EM Results 71
6.4.1. Basic Scenario 71
6.4.2. Five Target Scenario 74
6.4.3. LANL Scenario 76
6.4.4. Vertical Clutter Scenario 77

6.4.5. Truck Scenario 79
6.5 EM Approximate Median Results 81
6.5.1. Basic Scenario 81
6.5.2. Five Target Scenario 83
6.5.3. LANL Scenario 85
6.5.4. Vertical Clutter Scenario 86
6.5.5. Truck Scenario 88
6.6 Analysis of Results 90
6.7 Algorithms 91

Chapter 7 Conclusions and Future Work 93
7.1 Summary 93
7.2 Future Work 94
7.2.1. Real Time EM 94
7.2.2. POCA/EM Integration 95
7.2.3. Other Work 96

References 97

List of Figures

2.1 Illustration of cosmic ray shower cascade 7
2.2 Flux of cosmic ray particles at different altitudes 8
2.3 Energy distribution of cosmic ray muons at sea level 9
2.4 Mean energy loss of muons through different materials 10
2.5 Plot of Moliere's Au foil experiment 12
2.6 Chart of scattering amounts of muons through different materials 13
2.7 Drift tube chamber at LANL 14
2.8 GEM detector at Fl. Tech 15
2.9 Illustration of the muon tomography concept 16
4.1 POCA Concept 26
4.2 Plot of the points of closest approach 28
4.3 Parameters for 3D adjustment in EM 32
4.4 Illustration of the scattering and displacement of a muon through
many layers of material 34
4.5 Example of approximate median using binning 39
5.1 Simple Geant4 scenario 44
5.2 CRY produced muon spectrum at sea level 45
5.3 Illustration of the EM data structure 50
5.4 Illustration of the voxel structure 51
5.5 Plot of POCA scattering angles versus scattering angles from Geant4 57
6.1 Geometry of the basic scenario 61
6.2 Geometry of the five target scenario 63
6.3 Geometry of the LANL scenario 64
6.4 Geometry of the vertical clutter scenario 65
6.5 Geometry of the truck scenario 66
6.6 POCA results for the basic scenario 67
6.7 POCA results for the five target scenario 68

6.8 POCA results for the LANL scenario 69
6.9 POCA results for the vertical clutter scenario 69
6.10 POCA results for the truck scenario 70
6.11 Average-EM results for the basic scenario 72
6.12 Average-EM results for the five target scenario 74
6.13 Average-EM results for the LANL scenario 76
6.14 Average-EM results for the vertical clutter scenario 77
6.15 Average-EM lego plots for the vertical clutter scenario 78
6.16 Average-EM results for the truck scenario 80
6.17 Median-EM results for the basic scenario 82
6.18 Median-EM results for the five target scenario 84
6.19 Median-EM results for the LANL scenario 85
6.20 Median-EM results for the vertical clutter scenario 86
6.21 Median-EM lego plots for the vertical clutter scenario 87
6.22 Median-EM results for the truck scenario 88

List of Tables

6.1 Average-EM threshold analysis for the basic scenario 73

6.2 Average-EM threshold analysis for the five target scenario 75

6.3 Average-EM threshold analysis for the LANL scenario 76

6.4 Average-EM threshold analysis for the truck scenario 80

6.5 Median-EM threshold analysis for the basic scenario 83

6.6 Median-EM threshold analysis for the five target scenario 83

6.7 Median-EM threshold analysis for the LANL scenario 85

6.8 Median-EM threshold analysis for the truck scenario 89

6.9 Timing comparison for the average and median EM 92

Acknowledgements

This thesis would not have been completed if not for the support and help of many people. First I'd like to thank the U.S. Department of Homeland Security for their support and giving me the opportunity to continuing my education and enage in exciting research to help protect our country. This was done under grant 2007-DN-077-ER0006-02.

Dr. Debasis Mitra first asked me to join this effort several years ago while I was still an undergraduate. Through our work together we have had some great experiences that I will always remember. If not for his guidance and help I could not have made it through and I'll always be grateful for that. I would also like to thank Dr. Shoaff and the C.S. Department for their support and helping to send to me to Taiwan to present our work at the 2009 IEA-AIE conference.

Dr. Hohlmann was another great influence on me and my work and it's been wonderful working for him for the past several years. His constant attention to detail and improvement heavily shaped my work and work ethic. I will always appreciate my time working in the HEP lab with him.

Many others provided immense help throughout my research. David Pena, Jennifer Helsby, Patrick Ford, Kondo Gnanvo, Sammy Waweru, and several others guided me in working through the intricacies of Geant4 and using Linux systems, as well as shared in showing me the joy that is ROOT programming. Without their help and friendship my experience at Fl. Tech would have bee much less fulfilling. Thank you all.

Thanks goes to the whole Computer Science department who helped shape me as a student and as a person. Dr. Ford has influenced me the most in how I code and solve problems. Rosalyn Bursey has been a super woman making sure I could graduate without getting a migraine and helped solve any problems I ever encountered. Dr. Shoaff has been always willing to help out when needed as well. The Florida Tech Computer Science Department is second to none and I will

always cherish my time spent here.

Last but not least I'd like to thank my parents. They have been the most important figures in my life and guided and provided for me. Without their support I would never have even attended college let alone do post graduate work. Thank you for everything you've done and I can't wait to pay you back for all that you've given me.

Dedication

This thesis is dedicated to my parents. Without their loving support none of this would have been possible. Thank you.

Chapter 1

Introduction

1.1 Purpose of Study

Muon tomography is a relatively new type of imaging process making use of high energy particles. There are many different techniques involving the use of high energy particles that can be used for imaging. This study was based upon the use of cosmic ray muon particles to inspect cargo containers for nuclear material. Over six million shipping containers enter US ports each year and not even 5% are inspected manually or using some type of imaging [1]. Even more containers and vehicles enter the country through roads, rail and air that are never inspected. The development of a cost and time effective method to inspect these containers would greatly reduce the risk of dangerous materials being smuggled into the country. The purpose of this study was to implement and confirm known algorithms used in

muon tomography, as well as to develop improvements for them.

1.1.1 What Is Muon Tomography?

Muon tomography is an outgrowth of emission tomography which has been used for many years, especially for medical applications [2]. Muons are elementary particles that are similar to, but much more massive than, electrons. Most muons reaching the Earth come from cosmic rays. These high energy cosmic rays strike the atmosphere and produce a shower of particles, one of which is the muon. Due to their large mass, high energy muons can pass through many meters of material without being absorbed, and because of their high energy, are also easily detectable. The basic idea of muon tomography is to detect a muon before and after it travels through a volume that is to be imaged. Based on information measured and inferred from these tracks, a 3D image can be produced as well as other types of analysis to estimate what is inside. Reconstruction algorithms are needed to process this information, which is the focus of this study.

1.1.2 Importance of Muon Tomography

There are several different ways to try to image and/or detect radioactive materials in a volume. What are the advantages of muon tomography over these other methods?

Two methods the Department of Homeland Security is currently using are gamma ray radiography and passive gamma ray detection [1], both of which have some serious limitations and hazards. Gamma ray radiography makes use of gamma rays from radioactive isotopes of certain elements. They pass through a volume and through the attenuation of rays an image can be created. There is a safety issue as an artificial source of radiation needs to be introduced. Much care has to be taken to make sure humans don't get in contact with the harmful gamma rays. Besides the

safety hazards, gamma rays have limitations. They are not very penetrating, so using lead or other dense material to shield nuclear material could prevent the gamma rays from detecting the material. This makes hiding the presence of nuclear materials in containers a real possibility.

Passive gamma ray detectors are widely used at border crossings and other areas. They work by detecting the gamma rays that are produced by radioactive material. With many of these types of detectors the false alarm rates are very high due to their detecting sources of low-radiation that are used in general commerce for non-nefarious purposes. More sophisticated detectors can account for this, such as germanium based detectors, but have their own problems, like having to be operated at cryogenic temperatures. Also, proper shielding stops gamma rays from escaping these radioactive elements preventing them from being detected.

Other methods for detecting radioactive materials include using x-rays, like in a CAT (Computerized Axial Tomography) scan. Similar to gamma ray radiography though, artificial radiation sources are needed. These are both costly and expensive. The amount of energy required to successfully image the area needed for shipping containers is extremely large and because of the shielding needed for safety purposes as well as the energy costs, these systems could reach upwards of $100 million dollars [1]. Another problem with methods like these is that they rely on the skill of the person operating the systems to determine if something unusual is in the volume. This is not ideal as it opens the possibility for human error.

This is a general overview of the main techniques used for volume imaging as well as detection of radioactive materials. Other techniques exist but share much of the same problems as the ones discussed [1]. Muon tomography avoids most of the problems inherent to the previous techniques. Since the muons being used are from

cosmic rays, there is a natural source of radiation ready to be used and a potentially dangerous, artificial source need not be introduced. The high energy of muons allow them to pass through most shielding without being absorbed, thus avoiding the problem of gamma ray detectors where the rays are attenuated. Even if enough shielding were used to absorb most muons, the lower muon flux itself would be an indicator of shielding or large presence of very dense material which would be a warning flag in and of itself. Also, the methods for muon tomography (as will be seen later) do not rely on the skill of the operator to determine whether a dangerous material is present in a volume. Based on the information that can be obtained from the muon detectors and the way the reconstruction algorithms use them, a good estimate can be made of what type of materials are contained in the area being imaged. All these factors also make stations used for muon tomography potentially less expensive than the previously mentioned methods. For these reasons, muon tomography is a very attractive method for cargo inspection.

1.2 Scope of Study

This study covers some of the reconstruction algorithms used for muon tomography. The main focus is on the validation of these algorithms as well as possible improvements. Much background is also provided in this thesis on muons, tomography, as well as different tools used in the research. The results shown in this paper though are based on simulations and the brunt of the work is focused on the software engineering side of these algorithms as opposed to the theory behind them.

1.3 Thesis Outline

In Chapter 2 the necessary background information is provided to explain the concept of muon tomography, how and why it works, as well as past work involving its use. Chapter 3 delves into important questions about muon

tomography and what the goals of the work are. Chapter 4 takes an in depth look into the reconstruction algorithms used and explains how and why they work. Chapter 5 looks into the implementation of the reconstruction algorithms, as well as other tools created for simulation and analysis, and finally the software testing techniques used for debugging. Chapter 7 discusses the simulations done and the results obtained from them. Chapter 8 presents conclusions based on results obtained, as well as a look into future work.

Chapter 2
Background Information

2.1 Introduction

Chapter 2 relates important information about cosmic ray muons and other information that is important to know in order to understand muon tomography. Section 2.2 explores the general category of tomography and what it is used for. Section 2.3 gives a detailed description of muons and the physics relevant to them and this study. Section 2.4 combines the information from the previous two sections to illustrate the muon tomography concept. Section 2.5 concludes the chapter by taking a look at past work involving muon and emission tomography.

2.2 Tomography

Tomography can be defined as imaging by sections. One of the first uses of tomography was using X-rays to get images of areas inside the human body by moving the X-ray source and film in differing directions. Today there are many varied uses for tomography in a wide array of fields. Medical imaging still makes much use of tomography [3][4][2] with techniques now common to most of us like PET (Positron emission tomography), SPECT (Single photon emission computed tomography), and CT (Computed Tomography) scans. Other fields making use of tomography range from archeology for non-invasive surveying of ancient ruins [5], to geophysics which uses seismic tomography to estimate what the inside of the earth looks like, and even oceanography which makes use of sound waves from different projections to model objects in the ocean [6]. Many other types of tomography exists, but modern tomography generally involves gathering

projections from many directions and using a reconstruction algorithm to produce the results desired (i.e. a 3D image). These algorithms are a large area of research in tomography, as they can be very computationally expensive, and the balance between time and accuracy becomes a large issue. Muon tomography is a unique new method, but the issue of time and accuracy in reconstruction algorithms still remains.

2.3 Muons

Almost all naturally occurring muons on the earth are produced from cosmic rays. These are the muons that are of interest to muon tomography as they do not require an artificial source and have several attributes that can be taken advantage of. These attributes will be looked at in this section.

2.3.1 Basics

Muons are elementary particles similar to an electron from the lepton family of particles. They may be positively or negatively charged. Muons have a rest mass of 105.7 MeV/c^2 [7], which makes them almost 200 times more massive than electrons. They have a lifetime of 2.2μs which makes it the second longest living unstable subatomic particle, behind the neutron which has a lifetime of approximately 15 minutes [7].

The previous two facts about muons play a large role in their use for tomographic applications. Since muons are so massive, they don't give off as much electromagnetic radiation while traveling through matter as lighter particles do. This allows them to penetrate many meters of material before fully giving off their energy and being absorbed, which is important if some volume is to be probed. Also, since the muons being used are from cosmic rays, their relatively long lifetime is important as well, as most particles produced from cosmic ray decay are

short lived and don't reach the surface of the Earth. This will be explored in the next section.

2.3.2 Cosmic Ray Muons

The Earth is constantly being bombarded by high energy particles originating outside of our solar system. These particles (we will call them primary) are generally stable and when they strike our atmosphere interactions occur that produce other secondary particles, which in turn interact with the atmosphere and produce more particles themselves. This process is illustrated in Figure 2.1.

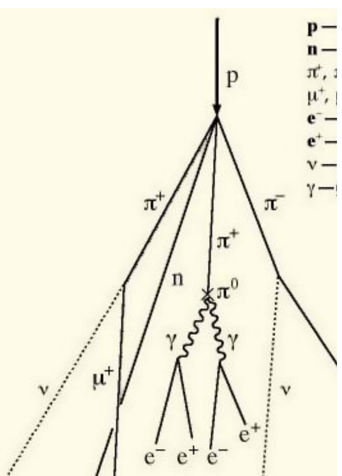


Figure 2.1: Cosmic Ray Shower Cascade [8]

The primary particles, usually highly energetic protons, enter the atmosphere and interact with other atmospheric nuclei and produce secondary particles called pions. Pions are short lived and do not usually reach the Earth before decaying into other particles. Charged pions decay into charged muons and the neutral ones decay into gamma rays, which may convert into electrons and positrons. Many of these particles lose their energy and dissipate in the atmosphere. However, most cosmic ray muons have sufficient energy to reach the Earth and since they have a longer

lifetime than the other particles produced in these showers, they are the most prevalent particles seen at sea level. This is illustrated by Figure 2.2, which shows the intensity fluxes of different cosmic ray particles at varying altitudes.

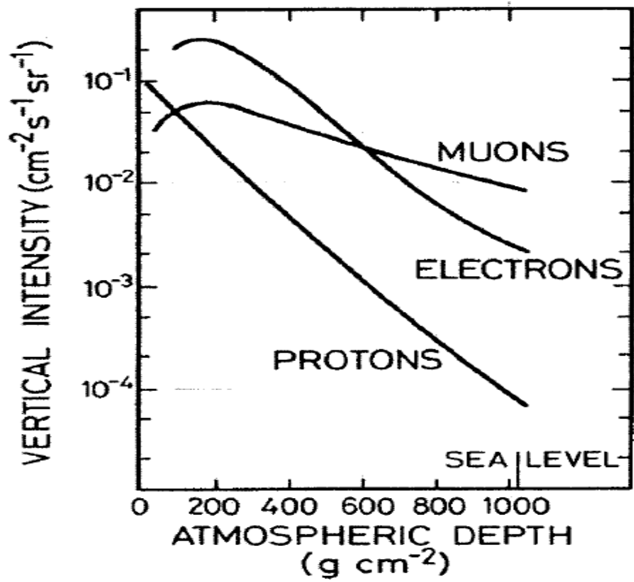


Figure 2.2: Flux of cosmic ray particles at different altitudes [9]

Muons arrive at the surface from a wide range of angles and energies. These distributions are dependent on many factors including how the particle is generated, the energy loss it experiences traveling through the atmosphere, as well as how the particle decays. The actual spectrum also is dependent on factors such as altitude, physical location on the Earth (for example the Earth's magnetic field filters out lower energy muons [10]), as well as solar activity which can alter the cosmic ray spectrum.

Many experiments have been done to measure the energy and angle distributions of muons. There are certain accepted tenets about cosmic ray muons though that experimentalists go by. They are listed here from the Review of Particle Physics by

the Particle Data Group [7]. The first is that the energy distribution for muons is relatively flat for energies lower than 1 GeV while it declines for those over 10 GeV, which results in the mean muon energy being between 3-4 GeV. This distribution is illustrated in Figure 2.3. Second, the muon flux is highest at the zenith and if θ is chosen to represent the angle between the muon path and the vertical then the drop off can be approximated as $\cos^2(\theta)$. Lastly, the muon flux at sea level for horizontally oriented detectors is about 1 per cm^2 per minute.

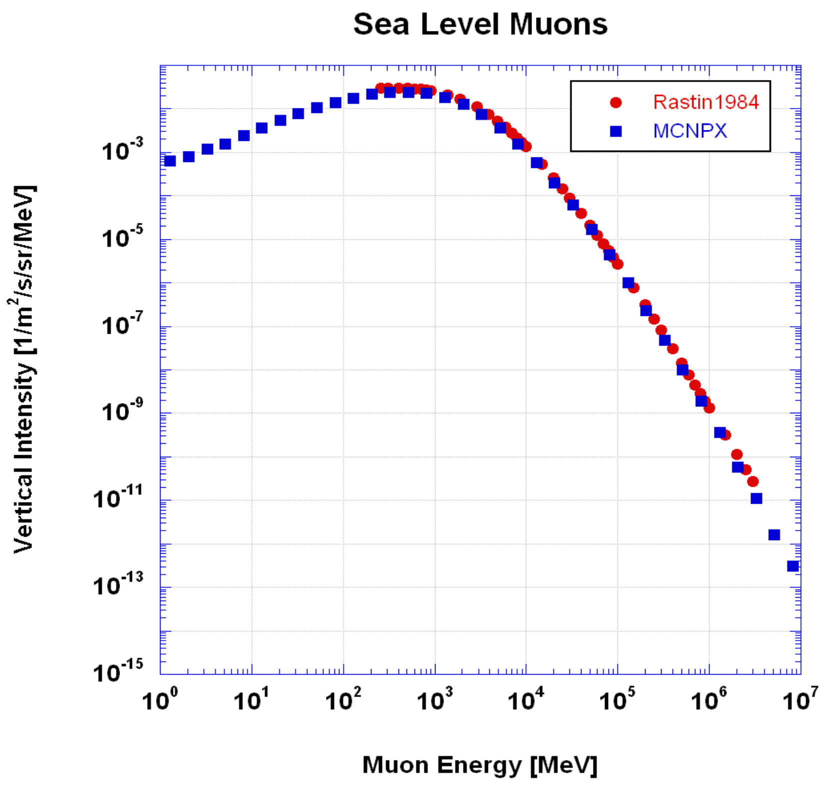


Figure 2.3: Energy distribution of cosmic ray muons at sea level [10]

2.3.3 Muon Physics

The way muons interact with matter is the prime reason they can be useful for the purposes of probing for certain materials. There are multiple ways muons are affected by their passage through some medium. As muons travel through atoms they can lose their energy through electromagnetic interactions. If enough energy is lost, the muon will stop. Muons are also diverted from their course as they pass atomic nuclei, which is called multiple Coulomb scattering. Both these interactions will be explained in more detail to show how they are relevant to muon tomography.

As muons pass through atoms they may strike electrons which are generally thrown out of their orbit and exit the atom. Based on the ionization energy of the electron, the muon will lose an equivalent amount. The amount of energy a muon loses passing through a material is dependent on the material itself, its thickness, as well as the momentum of the muon. Figure 2.4 shows the mean energy loss for muons

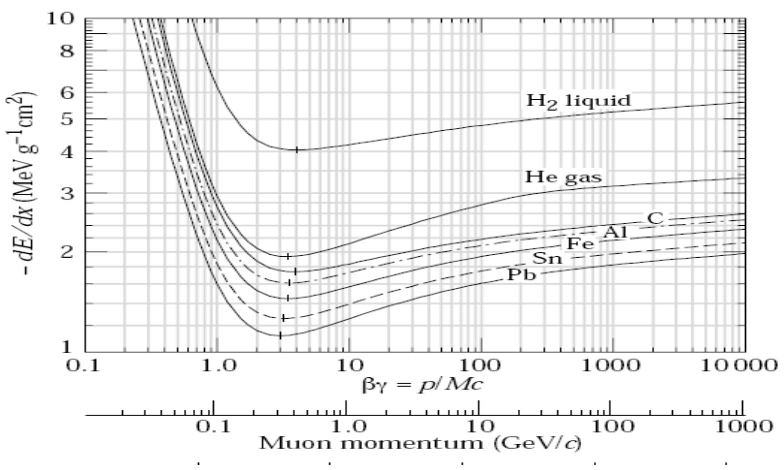


Figure 2.4: Mean energy loss (-dE/dx) for muons traveling through liquid hydrogen, gaseous helium, carbon, aluminum, iron, tin, and lead [7]

of varying momenta traveling through several different materials.

Using the energy loss as a means for scanning cargo is not currently a feasible strategy for imaging materials even though the energy loss does provide information on the material passed through. The issue is that precisely measuring muon energy is not a simple task and is unlikely to be a cost effective method [11].

When muons lose all energy they are absorbed into the material they pass through. A muons stoppage is also dependent on the type and depth of material, as well as its initial momentum. However, this feature of muons is not practical to use for cargo scanning either since muons are so penetrating. A cosmic ray muon at 3 GeV (the mean energy) can penetrate a meter of dense material, like lead or uranium, and higher energy muons can penetrate upwards of tens of meters of rock and medium density metals [11]. This would make measuring the loss of muon flux through only a few meters of material not very useful, which is the volume size of interest in cargo inspection. However, change of muon flux can be useful in imaging larger areas, as the spectrum of cosmic ray muon momenta is very wide and the loss of lower momentum muons passing through many meters of denser material would be apparent in comparison to a less dense (or open) area. In fact, some of the first attempts at using muons for imaging did exactly this and these attempts will be looked at in section 2.4.

As shown, energy loss does not provide a practical way to inspect small areas. Fortunately the second main interaction muons have while traversing matter could prove to be more useful. As muons pass near atomic nuclei their courses are altered from their paths many times. The scattering a muon experiences is non-deterministic. The distribution can be estimated as a zero mean Gaussian for the central 98% of the angles [7]. The width of the distribution can be defined as:

$$\theta_0 = \frac{13.6MeV}{\beta cp} z\sqrt{x/X_0}\left[1 + 0.038\ln(x/X_0)\right]$$

Here, p is the momentum, βc is the velocity, and z is the charge number of the incoming muon. The x/X_0 term is the thickness of the material being traversed in radiation lengths. For our purposes, β equals 1, and the particles are singly charged so z is equal to one as well. The value for θ_0 is actually a fit to the Moliere distribution for scattering angles [7]. There are many different theories for the theory of multiple scattering. Moliere's holds up well to experimentation while remaining transparent [5]. Figure 2.5 shows experiments validating how well Moliere's theory actually compares with real results.

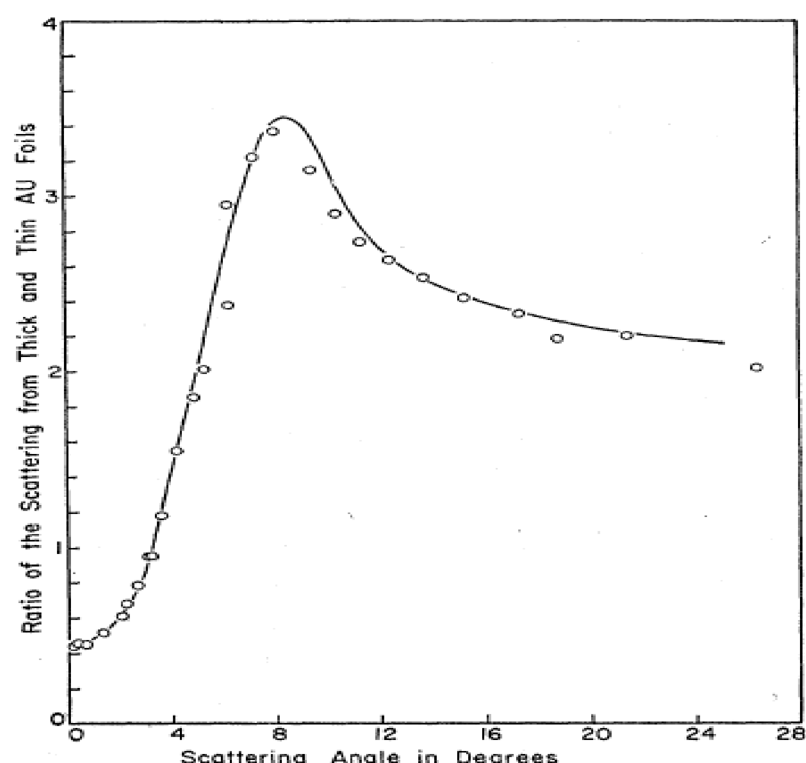


Figure 2.5: The line represents the predicted distribution of scattering angles by Moliere plotted against the ratio of thin and thick Au foils. The dots represent actual experimental result [12].

Radiation length (the units of the x/X_0 term) generally decreases as Z, the atomic number of an element, increases. This is important because radiation length is representative of how much matter there is for electromagnetic interactions. As the

radiation length increases, the angular distribution of the scattering widens. This is illustrated in Figure 2.6. Like energy loss, multiple Coulomb scattering is very sensitive to the material being traversed. Fortunately, unlike energy loss, muon detectors can precisely measure scattering more easily than energy loss. Thus this is the information that shall be made most use of in muon tomography.

Table of Mean Scattering Angle Spread for Various Materials

Material	Z	Density	Radiation Length	Weight of 10 cm diameter sphere	radiation lengths in 10 cm	Mean Scattering angle spread for 3 GeV muons
		(g/cm3)	(cm)	(kg)		(mrad)
Polyethylene	1 and 6	0.95	47.9	.5	0.21	2.44
Concrete	(Z/A=0.5)	2.3	11.55	1.2	0.86	5.25
Aluminum	13	2.7	8.9	1.4	1.12	6.05
Iron	26	7.87	1.76	4.1	5.68	14.44
Zirconium	40	6.51	1.56	3.4	6.41	15.4
Copper	29	8.96	1.43	4.69	6.99	16.13
Lead	82	11.35	0.56	5.94	17.86	26.64
Tungsten	74	19.3	0.35	10.1	28.57	34.24
Uranium	92	18.95	0.316	9.92	31.64	35.91
Plutonium	94	19.8	0.299	10.36	34.48	37.85

Figure 2.6: Amount of scattering for a muon passing through 10cm of different materials [1]

2.3.4 Muon Detectors

As described in the last section, there are two types of information muons can be mined for: energy loss and scattering. Scattering was said to be much easier to determine precisely. This section will explain how detecting the muons is actually done. There are several types of detectors. This is not a comprehensive overview, but will briefly explain about how some of the different detectors work, as well as the gas electron multiplier (GEM) detectors that will be used in our projects. The advantages and disadvantages of each will also be explained.

Drift tubes are a type of gas detector. They are cylinders made of some light metal with a thin wire stretching through the center. When a muon enters the tube it

ionizes the gas, which leads to an avalanche that ends when the electrons from the ionization reach the center wire. Based on the time it takes for the electrons to drift to the wire, the position of the muon can be determined [13]. These are the type of detectors used by Los Alamos National Laboratory [14]. Although drift tube detectors are a proven technology, because they rely on timing information their maximum precision is not as high as some other detectors. Figure 2.7 shows the drift tube detector system used at Los Alamos National Laboratory.

Figure 2.7: Drift tube chambers used at Los Alamos National Laboratory [14]

GEM detectors [15] are a newer type of detector relative to drift tubes in the category of micropattern gas chambers. Generally gas chamber detectors amplify the electrons knocked out of a gas by charged particles as they pass through. Unlike drift tubes, which rely on timing information after the gas is ionized, micropattern detectors measure the spatial coordinates of where the electron avalanche reaches the signal induction strip. The distance between these sensitive elements can be

reduced to a fractions of millimeters based on photolithography [16] allowing measurements with very high precision. GEM detectors improve upon this performance by using a thin sheet of plastic coated with metal on both sides on which tiny holes are bored into only microns apart. Applying a voltage across the foil causes an avalanche of ions and electrons pour through the holes [15]. Then the typical process of gas chambers with micropattern readout takes over as was described before. Since spatial information is being used rather than timing, the overall resolution can be about 50 microns [17][16]. In the High Energy Physics lab at the Florida Institute of Technology, small prototypes of muon tomography systems employing GEM detectors are being developed [17], and the reconstruction algorithms being developed in this study will receive their input data from these detectors when they are complete instead of getting them from simulations as is our current practice. Figure 2.8 shows the first GEM detector built by the HEP lab at Florida Tech.

Figure 2.8: GEM detector constructed by HEP Lab at Florida Tech

2.4 Muon Tomography

Now that the relevant physics applying to muons is described, as well as the way to detect them, we have the background information to explain muon tomography.

2.4.1 Concept

The idea is to put the volume to be probed between two sets of detectors (or more, if lateral detectors are added), detect muon events for a certain amount of time, determine the scattering angle between the tracks as well as other useful information (which will be discussed later), and then pass the information to algorithms to reconstruct it into a 3D image or some other form of output. Figure 2.9 displays the concept.

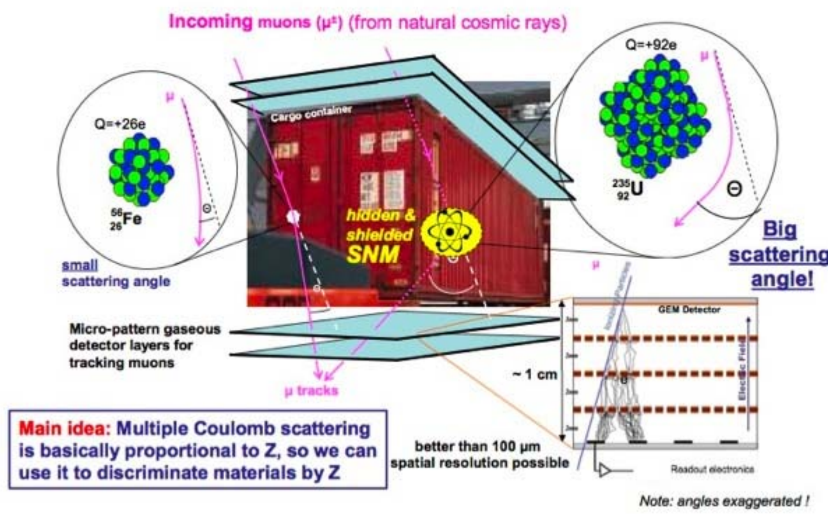


Figure 2.9: The muon tomography concept. A muon passing through uranium has the potential to scatter more than one that passes through iron. [18]

2.4.1 Reconstruction Algorithms

By the combination of the scattering information with other measurable information, assumptions, and heuristics, reconstruction algorithms can be developed for producing 3D images of the area being scanned, as well as for discriminating between different materials. Some reconstruction algorithms for muon tomography will be discussed in section 2.5, but the ones explored mainly in this thesis are the Point of Closest Approach algorithm and an Expectation Maximization algorithm created by Los Alamos National Laboratory [19]. These will be looked at in depth in chapter 3.

2.5 Past Work

Muon tomography is a relatively new type of imaging technique, although the methods themselves have been around for quite some time. Muons have been used for imaging large structures for several decades. Emission tomography has been around since the 1980's when Positron Emission Tomography was introduced. From these methods, as well as a couple other novel ideas, sprung the concept of muon tomography. These past attempts will be explored in more detail, starting with muon radiography.

2.5.1 Muon Radiography

Because of the energy spectrum of cosmic ray muons being well established, measuring the muon flux before and after they pass through a large mass could provide much information about the objects traversed since the change in muon flux would relate directly to the material and its depth. Several researchers have taken advantage of this technique.

The first to do so was E.P. George in 1955 [20]. He wanted to measure the depth of an underground tunnel, so he measured the cosmic ray muon flux inside and outside the tunnel. Based on the ratios of the attenuation of the muon flux he made

a good estimate of the depth of the tunnel. A similar and more famous example of this technique was used by Luis Alvarez in the 1960's [21]. What Alvarez intended to do was use muon radiography to determine if there were any hidden chambers in the Pyramid of Chepren at Giza, as there were many in the pyramids of both Chepren's father and grandfather. By placing muon counters in already discovered hidden chambers, Alvarez looked to match the muon flux measured at different angles with the muon flux that would be expected from the depth of rock being traveled through. Based on the information gathered he was able to confirm that no other hidden chambers existed in the pyramid.

Many researchers have followed similar approaches. An ambitious attempt was made by Nagamine [22] to predict whether or not Mt. Tsukaba and Mt. Asama in Japan would erupt. He used large detectors oriented horizontally (this was because large angle muons with almost flat trajectories were needed) spaced 2 kilometers apart on the sides of the mountains to image the internal structure of the volcanoes. In another attempt at making use of muons for imaging, Minato [23] managed to radiograph Higashi-Honganji Temple Gate in Nagoya, Japan, armed with only a hand held muon counter.

A different approach using muons for radiography was made by Frlez and his colleagues [24]. Their interest was in measuring how efficient cesium iodide crystals were for calorimetry. Several crystals were placed between muon detectors and the rays were measured as they passed through the volume and the crystals. For the tracks that went through the crystals, the path length and energy loss in the crystal were measured. Based on this, the energy deposition into the crystals could be analyzed and the efficiency of the crystals determined.

All these previous examples were novel ways to use muons to probe volumes using

radiography, but none made use of multiple Coulomb scattering to image smaller areas. Past attempts at using this information source will be looked at next.

2.5.2 Muon Tomography

The first attempt at using multiple Coulomb scattering of muons for homeland security purposes was started in 2001 by a group at Los Alamos National Laboratory (LANL) [11]. A description of their early work and results with simple scenarios was published in an issue of the science magazine Nature in 2003 [25]. Here a tungsten cylinder was reconstructed using a simple reconstruction algorithm. Continuing research has brought about more sophisticated algorithms and muon tomography systems the past several years.

Part of the original group was Larry J. Schultz who went on to develop a much more in depth algorithm detailed in his dissertation [11]. In it he describes a maximum likelihood algorithm (a method of fitting statistical data to a model) based on the scattering and displacement of muons. The results produced by the algorithm were very good, though the computation time and memory usage posed major issues for use in larger, more realistic scenarios [11][19]. The original LANL group continued work on this maximum likelihood algorithm, improving its robustness and running time while also developing a small prototype muon tomography system [26][27][28]. Eventually an expectation maximization algorithm (this type of algorithm will be looked at more closely in the next section and chapter 4) was created by Dr. Schultz and the LANL group, which improved the computational performance of finding the maximum likelihood estimates as well as the handling of the non-Gaussian scattering of muons [19]. This is the approach that the reconstruction algorithms in this study are based upon.

Another technique was proposed by Dr. Wang and Dr. Qi from the University of

California, Davis. While the LANL approach did not involve modeling the non-Gaussian scattering of muons, Wang and Qi fully modeled the scattering distribution of muons by using a Gaussian scale mixture (a combination of multiple probabilistic models). They also used maximum likelihood estimates for reconstruction based on the mixture model created, but Bayesian statistics were also incorporated to increase the quality of the results [29].

Other attempts have been made to verify results seen in literature based on existing algorithms, like the simple, geometry based reconstruction algorithm, Point of Closest Approach (POCA). Gnanvo, et al., [17] have used Geant4 simulations along with POCA to show that discrimination of different materials using muon tomography is feasible with a high enough detector resolution. C. Motooka and Y. Watanabe have also experimented with the POCA algorithm and concluded that cosmic ray muon tomography is a viable way to discriminate between materials [30]. This author has done work with Dr. Debasis Mitra and Dr. Marcus Hohlmann, coming to similar conclusions about cosmic ray muon tomography being a real possibility for cargo inspection based. This is based upon on results seen from POCA and the EM algorithm developed at LANL [31]. In addition to the reconstruction algorithms, companies such as Mu-Vision have plans to construct portable muon tomography stations that can be easily deployed and taken apart for use at ports or border crossings [1].

This section has displayed that the use of muon tomography for material discrimination is well founded, based on the results from many disparate groups. Although not making use of muons, emission tomography has been heavily used in medical applications, and many of the techniques are also applicable to muon tomography. These types of methods will be looked at next.

2.5.3 Emission Tomography

Medical imaging has made great use of emission tomography to help diagnose patients over the past couple decades. One of the first of these was positron emission tomography (PET), developed by Vardi, Shepp and Kaufman in the early 1980's [2]. The idea is to place positron emitting material into some organ to be imaged. When the positron is emitted and it strikes an electron both are annihilated and two X-ray photons are created traveling in opposite directions. Cylindrical detectors are placed around the patients body (usually the head) and the number of photons detected is counted. Pinpointing where the positron came from is impossible, but based on the density of photon emissions at different angles, a mathematical model can be developed. It is assumed that the emissions occur as a spatial Poisson process [2]. Based on the emission density and Poisson model, a maximum likelihood method using an expectation maximization (EM) algorithm (the general class of EM algorithms will be looked at more thoroughly in chapter 4) is used to reconstruct the 3D cylindrical image of the organ being looked at. They also developed a method of moments (a method of estimation of population parameters like mean, variance, median, etc.) and a least squares reconstruction. This work heavily influenced much emission tomography to follow and provided a huge contribution to statistics. Today, the EM algorithm is one of the most used methods for reconstruction in emission tomography.

The EM algorithm has also been used successfully in Single Photon Emission Computed Tomography (SPECT), which uses gamma ray emissions for its probe, as well as X-ray computed tomography (CT) [3]. These scans typically make use of filtered back projection (FBP) techniques for reconstruction, which is based on the Fourier Slice Theorem and makes use of the Fourier transform, making them fast and reliable. However, with the advancement of non-radon based scanners,

problems have arisen with FBP techniques that statistical algorithms like EM can better handle [3]. EM is not always the best choice of reconstruction algorithm for emission tomography, but it is continually being used more frequently and has done well when tested against competing algorithms [32].

Since EM has provided such good results in terms of accuracy and robustness, it has also been the area of much research, and many improvements and variations for it exist. For SPECT scans, the EM algorithm has been modified to incorporate Bayesian statistics to smooth out the results and has been shown to be successful by Green [3] and has been improved upon itself using other novel ideas like Markov random fields and Gibb functions by Herbert and Leahy [33]. Hudson and Larkin [34] developed an ordered subsets (OS) EM algorithm that breaks the input data into subsets, runs an iteration on a set, and feeds that data to the next subset until all are processed. This sped up computation immensely while still producing acceptable results. This OS-EM was shown to be useful for both PET and SPECT scans, but the concept itself can be applied to many different versions of EM and ML methods [34].   More recently, Sangtae Ahn, et al., developed an OS-EM algorithm that converges faster (previous OS-EM's did not converge and had to be arbitrarily stopped) than normal EM without loss in accuracy [35].

Emission tomography has enjoyed much success in the medical field, and has been used to diagnose and treat a multitude of illnesses. Besides the actual benefits though, the research done into this area has a lot of application in other fields, which made studying tomography a target topic of this thesis.

Chapter 3
Research Question

3.1 Research Questions

Although muon tomography is a relatively new type of technique used for cargo inspection, much research has been done in the area. Many different approaches have been used to develop reconstruction algorithms that are accurate and run in a reasonable amount of time. Balancing these criteria is essential in creating a useful reconstruction algorithm. This leads to the purpose of this chapter which is to define the goal of this study, the expected results, and the potential advantages this approach has over past approaches.

3.1.1 What is the goal of this work?

There are several goals this study intends to accomplish.

3. Confirm previous results from current algorithms, namely POCA and the EM algorithm proposed by LANL.
4. Propose an improvement to the median version of the EM algorithm
5. Do a detailed analysis into the accuracy of the various EM methods
6. Analyze the varying run times of the algorithms

Although these are the main goals of the work, other goals were met in the course of research. These will be brought up in the appropriate sections and discussed for their merit and why they didn't end up being part of the main focus of the project.

3.1.2 What are the expected or wanted results?

The overriding conclusion based on results from past work was that cosmic ray

muon tomography is definitely a practical way to unobtrusively probe unknown volumes. Desired results from this study would lead to the same conclusion. More specifically, the results should show the ability of the algorithms to discriminate between different materials. Since this has already been found by other groups, these results are also expected. What this study looks to add to the field is improvements on the existing algorithms (mainly the EM), so that they run faster and more efficiently, without sacrificing their ability to discriminate materials.

3.1.3 What are the potential advantages of this approach over others?

Expectation maximization algorithms in general take a long time to run in comparison to non-statistical reconstruction algorithms like the filtered back projection techniques that were briefly described in chapter 2. This is the case for the LANL EM algorithm as well. Run times of the EM algorithm have been acceptable for reasonably sized jobs. For large scenarios with lots of statistics though, reconstructions can take very long times (upwards of 12 hours in our simulations), making it impractical to test a scenario with many parameters. One aim of this study was to improve the runtime of the algorithm by removing unnecessary computation. This is one area where the approach detailed in this thesis can provide an advantage over existing techniques.

The memory issue is also extremely important because the resource requirement is so large that it limits the size of scenarios that can be run, as well as the number of muons that can be simulated. The High Energy Physics lab here at Florida Tech has a high performance computing cluster [57], yet several times scenarios were run that overwhelmed the system due to memory constraints. The aim will be to eliminate the memory waste inherent in the EM algorithm, as well as finding implementation techniques that further reduce the memory load. This would be another potential improvement over the other methods.

Chapter 4
Reconstruction Algorithms

4.1 Overview

In general there are two classes of reconstruction algorithms available for tomography: filtered backprojection (FBP) and iterative reconstruction (IR). Both types were briefly seen in section 2.5.3. The choice of algorithm mainly depends on the information source being used. FBP algorithms are relatively fast algorithms that make use of the fast Fourier transform, but need a set of evenly spaced projections around a wide area (usually at least 180 degrees) with straight ray paths [11]. This requirement makes FBP algorithms unsuitable for muon tomography as the cosmic ray muon angle spectrum is less than 180 degrees and not evenly spaced as was shown in chapter 2, and the tracks being used are not straight because of multiple Coulomb scattering. IR algorithms are algebraic and work by defining the volume being probed as a set of parameters. Sets of equations are created in terms of those parameters and the measured data (muon scattering in this case), and the reconstruction problem becomes solving these equations. The EM algorithm developed at LANL is one of these IR algorithms and will be explained in depth later in this chapter in section 4.3. First however, a more primitive, heuristic reconstruction algorithm that doesn't fit into either category is explored.

4.2 Point of Closest Approach

The Point of Closest Approach (POCA) is a geometrical algorithm with many applications. One of these is computer gaming where it is used heavily because of its usefulness for 3D imaging [37][38][39]. For muon tomography purposes, POCA is a heuristic algorithm that was developed by Los Alamos National Laboratory [32][25]. It was intended as a proof of concept algorithm to show the

possibilities of muon tomography. It has shown promising results and has been validated [17][30] as well as improved upon by different sources [11].

The concept of POCA is simple. It ignores multiple coulomb scattering and assumes a muon scattered at a single point. Based on projected the incoming and outgoing tracks, find the points where they came closest, estimate the scattering point as the midpoint of the line between the points of closest approach, and measure the angle between the incoming and outgoing tracks. The concept is illustrated in Figure 4.1.

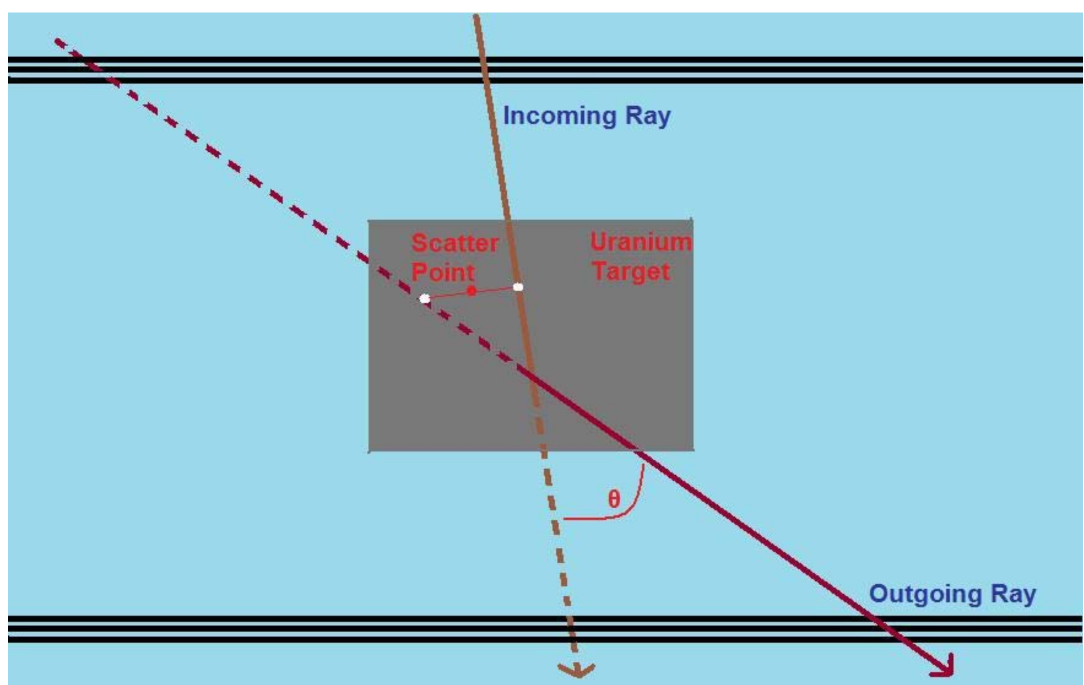


Figure 4.1: POCA Concept: Measure incoming and outgoing tracks and estimate where the muon scatters as the point where the tracks come closest

Many types of analysis can be done on the information gathered from POCA, but the general way is to plot the point and color it according to the magnitude of the scattering angle. Results from POCA will be shown in chapter 7. Next the POCA

algorithm used will be explained in detail.

4.2.1 POCA Algorithm

There are many ways to determine the points of closest approach. The goal is to find the shortest line between two vectors and there are ways to do this using calculus [38] or geometry [39]. The algorithm implemented in this study was developed by Dan Sunday [37]. It was chosen because it works in any dimension and is comparatively faster than other algorithms researched [38][39].

Consider two infinite lines (line $\mathbf{L}_1$ represented by P and line $\mathbf{L}_2$ represented by Q) using parametric equations, $\mathbf{L}_1$: $P(s) = P_0 + s\,(P_1 - P_0) = P_0 + s\mathbf{u}$ and $\mathbf{L}_2$: $Q(t) = Q_0 + t\,(Q_1 - Q_0) = Q_0 + t\mathbf{v}$. Create a vector between any points on the two lines, $\mathbf{w}(s,t) = P(s) - Q(t)$. $\mathbf{L}_1$ and $\mathbf{L}_2$ are closest at the unique points $P(s_c)$ and $Q(t_c)$ for which $\mathbf{w}(s_c,t_c)$ has its minimum length. Also, the line segment connecting $P(s_c)Q(t_c)$ is perpendicular to both lines at the same time and is the only line segment between the lines that has this property. In other terms, the vector $\mathbf{w}_c = \mathbf{w}(s_c,t_c)$ is perpendicular to the line direction vectors of $\mathbf{L}_1$ and $\mathbf{L}_2$, $\mathbf{u}$ and $\mathbf{v}$, and finding this vector is the same as solving the two equations: $\mathbf{u} \cdot \mathbf{w}_c = 0$ and $\mathbf{v} \cdot \mathbf{w}_c = 0$. Figure 4.2 graphically shows this concept.

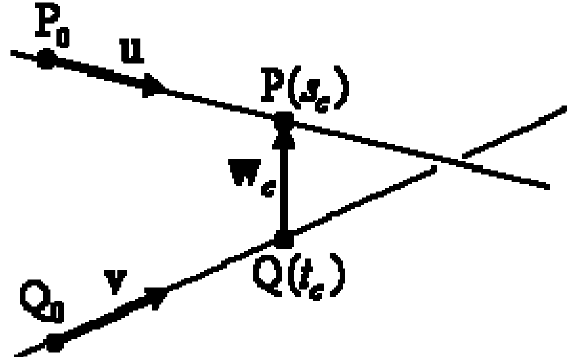


Figure 4.2: Points of closest approach (P(Sc), Q(Tc)) are the end points of the line segment where the length of Wc is at a minimum [40]

By substituting $\mathbf{w}_c = P(s_c) - Q(t_c) = \mathbf{w}_0 + s_c\mathbf{u} - t_c\mathbf{v}$, where $\mathbf{w}_0 = P_0 - Q_0$, into those two equations we can get two simultaneous linear equations so that the original two can be solved for: $(\mathbf{u}\cdot\mathbf{u})s_c - (\mathbf{u}\cdot\mathbf{v})t_c = -\mathbf{u}\cdot\mathbf{w}_0$ and $(\mathbf{v}\cdot\mathbf{u})s_c - (\mathbf{v}\cdot\mathbf{v})t_c = -\mathbf{v}\cdot\mathbf{w}_0$. Next algebra is used to solve for s_c and t_c. Let $a = \mathbf{u}\cdot\mathbf{u}$, $b = \mathbf{u}\cdot\mathbf{v}$, $c = \mathbf{v}\cdot\mathbf{v}$, $d = \mathbf{u}\cdot\mathbf{w}_0$, and $e = \mathbf{v}\cdot\mathbf{w}_0$. This gives: $s_c = be-cd / ac-b^2$ and $t_c = ae-bd / ac-b^2$. Whenever $ac-b^2=0$ the lines are parallel and all points are points of closest approach. How this is implemented will be explained in chapter 5. Now that s_c and t_c are solved for, the points of closest approach can be determined by using the original equations for the line, $P(s_c)$ and $Q(t_c)$.

4.2.2 POCA Analysis

As stated before, POCA was designed as a proof-of-principle algorithm and is not based on the actual physics of multiple coulomb scattering. In fact, POCA assumes a single scattering event. Thus, POCA should work best in scenarios where a muon experiences few closely spaced scattering events, which would be when a muon travels through limited amounts of material. When a muon travels through multiple objects, the scattering point would tend to be located between the objects as

opposed to a single point from either object. This would seem to limit POCA to cases where there is little material or few obstructions in the target volume. The results of POCA run on different scenarios will be shown in chapter 7.

POCA is a relatively quick algorithm in comparison to other statistical algorithms generally used in reconstruction. The running time of POCA is based strictly on the number of muon events as every event is processed sequentially and once the points of closest of approach are found the event is discarded. The algorithm thus runs in $O(n)$ time. The only memory usage is storing the information needed for a single muon event, so the memory usage is constant and is $O(1)$.

4.3 Expectation Maximization (EM)

In general, the EM algorithm is used to find maximum likelihood estimates (methods used to fit data to a statistical model) of parameters in some probabilistic model, where the model is dependent on some other parameters that can't be directly measured but only inferred. EM is an iterative method with two steps. The first step is the expectation (E) step that takes the current estimate of the model for the “hidden” parameters and computes an expectation of the log likelihood for it. Next comes the maximization (M) step, that computes the parameters which will maximize the expected log likelihood found on the previous step. These parameters are then passed back to the E step to determine the new distribution of the hidden parameters. These two steps are done until the parameters converge or for a predetermined number of iterations. This framework can be adapted for many uses as previously shown as long as the data can be described by some probabilistic model based on some estimated parameters. How this was done for an EM algorithm for muon tomography will be explained next.

4.3.1 EM Model

The EM algorithm used in this study was originally developed at Los Alamos National Laboratory by Larry Schultz, et al. [19]. It was based on earlier work by Larry Schultz in his dissertation where he created maximum likelihood estimates based on the scattering angle and ray displacement information [11]. The information displayed here on the EM algorithm is taken mostly from from the former, but all the information on the algorithm in this section is taken from these sources.

The width of the distribution of the central 98% of scattering angles was described as a function of the material it was passing through. This was taken from the *Review of Particle Physics* [7], but as was stated, many have developed a theory of multiple scattering. The scattering function used for this algorithm was a simpler one found by Bruno Rossi [34]:

$$\sigma_\theta \cong \frac{15MeV}{\beta cp}\sqrt{\frac{H}{L_{rad}}}$$

As in the equation from the *Review of Particle Physics*, p is the momentum, and βc (β=1) is the velocity. H represents the depth of the material being traversed, and L_{rad} is the radiation length of the material.

A scattering density function is introduced in terms of the material being traversed for a particular momentum. This function describes the mean square scattering angle a muon would go through after passing through a unit depth of this material. So for some nominal momentum p_0 in GeV, and some material described in terms of L_{rad} the scattering density function is

$$\lambda(L_{rad}) \equiv \left(\frac{15}{p_0}\right)^2 \frac{1}{L_{rad}},$$

with units in milliradians^2 per cm.

The variance of the scattering distribution can be described in terms of λ:

$$\sigma_\theta^2 = \lambda H p_r^2 ,$$

with

$$p_r^2 = \left(p_0 / p \right)^2 .$$

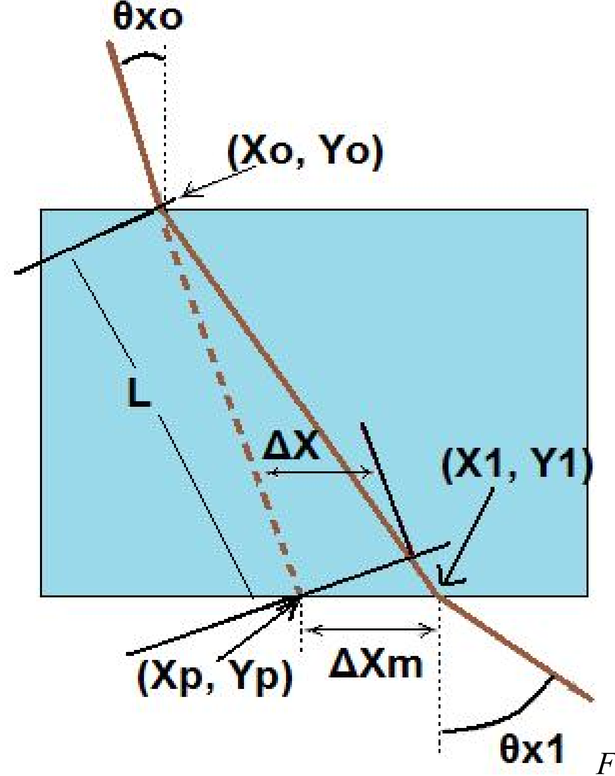
Besides the scattering angle of a muon, more information exists that can shed light on the muon's path through the volume. This is the displacement of the muon, which represents the distance between where a muon enters a volume and where an unscattered muon would have exited the volume. This is illustrated in Figure 4.3. It has been shown that the scattering and displacement are correlated and the distribution can be described as jointly Gaussian with zero mean [42] and the width's equated by:

$$\sigma_{\Delta x} = \frac{H}{\sqrt{3}} \sigma_{\Delta\theta}$$

The covariance matrix can be expressed by:

$$\Sigma \equiv \lambda \begin{bmatrix} H & H^2/2 \\ H^2/2 & H^3/3 \end{bmatrix} p_r^2 = \lambda A p_r^2$$

The amount of scattering and displacement gone through in the x and y directions are completely independent and distributed identically. For three dimensions the algorithm uses information broken down into these two directions, represented by $\Delta\theta_x$ and $\Delta\theta_y$ for the scattering, and Δ_x and Δ_y for the displacement. Figure 4.3 shows this information for one direction as well as other important parameters related to a muon's path through a single layer of material.



F

igure 4.3: Parameters used for 3D adjustment

Parameter *L* represents the estimated 3D path length of the muon through a layer of material. The actual path length is different because of the multiple coulomb scattering that goes on as the muon traverses the volume. This estimated path length can be found by the following equation using the initial incoming angles:

$$L = H\sqrt{1+\tan^2(\theta_{x0})+\tan^2(\theta_{y0})} \equiv HL_{xy}$$

Naively, by looking at figure 4.3 it would seem the displacement would be equal to $x_1 - x_p$, however this needs to be adjusted for 3D path length as well as be oriented in the proper direction. Thus the displacement should be defined as:

$$\Delta_x = (x_1 - x_p)\cos(\theta_{x0})L_{xy}\frac{\cos(\Delta\theta_x + \theta_{x0})}{\cos(\Delta\theta_x)},$$

where,

$$\Delta\theta_x = \theta_{x1} - \theta_{x0}$$

Schultz gives a derivation for this result [11].

Now instead of using *H*, the depth of a material passed through, the 3D path length through the material, *L*, can be used and the following redefined:

$$A \equiv \begin{bmatrix} L & L^2/2 \\ L^2/2 & L^3/3 \end{bmatrix}$$

The same process is used to determine scattering and displacement in the y direction. The rest of the algorithm will be described for only the x coordinate and then at the end y coordinate information will be introduced. It is noted as well that this model is valid only for small angle scattering [19].

A muon will travel through many different materials in a volume in a realistic situation, so a model needs to be developed that accounts for the many different layers a muon might pass through. This is accomplished by breaking the volume into many smaller rectangular sub-volumes called voxels and the scattering density for each will be attempted to be found. Now the hidden information for the EM algorithms comes into play. The amount of scattering or displacement a muon goes through a particular voxel cannot be measured directly. However, it can be described in terms of the measurable information. Figure 4.4 helps show the concept.

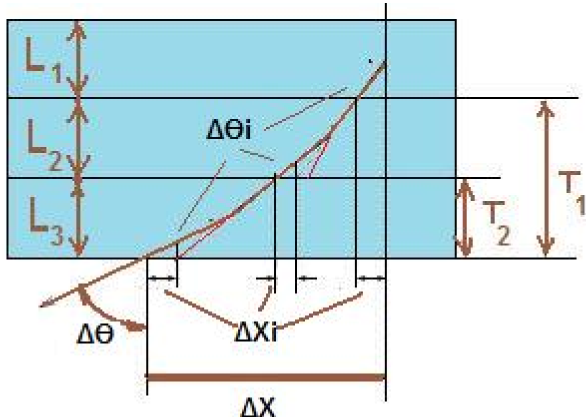


Figure 4.4: Scattering and displacement through many layers of material

The total scattering can be thought of as the sum of the scattering through each voxel with N representing the set of all voxels.

$$\Delta\theta = \sum_{j<N} \Delta\theta_j$$

Using the assumption of small angle scattering, the total displacement can be defined as the sum of the displacements as well as the sum of parameter T multiplied by the corresponding scattering angle in that voxel.

$$\Delta x = \sum_{j<N} \left(\Delta x_j + T_j \Delta\theta_j \right)$$

T represents the the 3D ray path-length from the exit point of a voxel to the exit point from the reconstruction volume. According to Schultz, T for a specific voxel can also be defined as the sum of all L values for the muon track after the voxel in question [11].

Let us set i as the number of muon rays and j as the number of voxels.
The covariance of aggregate scattering and displacement can be described for the ith muon for the jth voxel by [11]:

$$\Sigma_i = p_{r,i}^2 \sum_{j<N} \lambda_j \begin{bmatrix} w_{\theta,ij} & w_{\theta x,ij} \\ w_{\theta x,ij} & w_{x,ij} \end{bmatrix} = p_{r,i}^2 \sum_{j<N} \lambda_j W_{ij}$$

with the weights (the *w* parameters) defined by:

$$\begin{aligned} w_{\theta,ij} &= L_{ij} \\ w_{\theta x,ij} &= L_{ij}^2 / 2 + L_{ij} T_{ij} \\ w_{x,ij} &= L_{ij}^3 / 3 + L_{ij}^2 T_{ij} + L_{ij} T_{ij}^2 \end{aligned}$$

Where L_{ij} is the 3D path length of the ith muon through the jth voxel or zero if the muon does not pass through that voxel and N is the total number of voxels. Defining λ and the weights in terms of vectors, the covariance matrix can be expressed as:

$$\Sigma_i = \begin{bmatrix} \mathbf{w_{\theta,i}} \cdot \boldsymbol{\lambda} & \mathbf{w_{\theta x,i}} \cdot \boldsymbol{\lambda} \\ \mathbf{w_{\theta x,i}} \cdot \boldsymbol{\lambda} & \mathbf{w_{x,i}} \cdot \boldsymbol{\lambda} \end{bmatrix} p_{r,i}^2$$

The muon path through the volume must be estimated of course to obtain the previous information. Simply connecting the entering and exiting points of the muon is one alternative, but due to the multiple scattering a muon goes through this is obviously not a true representation of the path. We used POCA [19] as a way to estimate the muon's path through the volume by connecting the entering point to the scattering point and the scattering point to the exit point.

Lastly, let the data vector for all the measured muon events be defined by:

$$D_i \equiv \begin{bmatrix} \Delta\theta_i \\ \Delta x_i \end{bmatrix}$$

Thus the likelihood of a particular scattering density for a set of data can be expressed as:

$$P(D \mid \lambda) = \prod_{i \le M} P(D_i \mid \lambda),$$

with *M* representing all muon measurements and,

$$P(D_i \mid \lambda) = \frac{1}{2\pi|\Sigma_i|^{1/2}} \exp\left(-\frac{1}{2} D_i^T \Sigma_i^{-1} D_i\right),$$

where $P(D_i \mid \lambda)$ is the probability density function in terms of the measured data, D_i , and the scattering density, λ.

Thus finding an estimate for λ that maximizes the likelihood is the goal. Many different techniques exist for finding such a λ, but the ones explored by those at LANL did not work well for large, real life scenarios [19]. Thus an EM approach was developed. This is described in the next section.

4.3.2 EM Development

The EM algorithm begins with an auxiliary function which is to be maximized. For the EM algorithm for muon tomography the auxiliary function used is:

$$Q(\lambda; \lambda^{(n)}) = \mathrm{E}_{H|D,\lambda^{(n)}}\left[\log(P(H \mid \lambda))\right]$$

Here E is the expected value of the log likelihood of both the hidden and observed data. After some advanced statistical analysis and derivation(which is beyond the scope of this study), the expectation step can be defined as:

$$C_{ij} \equiv \left(D_i^T \Sigma_i^{-1} W_{ij} \Sigma_i^{-1} D_i - Trace\left(\Sigma_i^{-1} W_{ij}\right)\right) p_{r,i}^2$$

This expectation step can then be substituted into the update equation which is also the maximization step:

$$\lambda_j^{(n=1)} = \lambda_j^{(n)} + \left(\lambda_j^{(n)}\right)^2 \frac{1}{2|M_j|} \sum_{i:L_{ij} \neq 0} C_{ij}^{(n)}$$

Here M_j represents the number of rays that traveled through the jth voxel. The results of the expectation step are called correction values. The average correction value for all muons that pass through the jth voxel is used to update λ. The new λ values can then be passed back to the expectation step and used to compute the new correction values. The algorithm in general is displayed in the next section.

Since the model behind the EM was described for only 98% of of the scattering distribution, results where λ values were too large occurred [19]. This is because 2% of the angles scattered more than the model accounts for. Since the mean of the correction values is used to update λ, the larger than expected scattering angles causes the mean to come out too large. To account for the data that doesn't fit into the Gaussian model, a median correction value is used as opposed to the mean:

$$\lambda_j^{(n=1)} = \lambda_j^{(n)} + \left(\lambda_j^{(n)}\right)^2 median\left(\sum_{i:L_{ij}\neq 0} C_{ij}^{(n)}\right)$$

4.3.3 EM Algorithm

Input: Scattering and displacement in x and y of each muons as well as the momentum parameter for each muon ($\Delta\theta_x$, $\Delta\theta_y$, Δ_x, Δ_y, p_r^2)

Estimate (L,T) for every voxel of every muon track;
Compute weights (w_θ, $w_{\theta x}$, w_x) for every voxel of every muon track
Initialize for each voxel;
Set max iterations *I;*

(1) for each iteration k = 1 to I do
 (2) for each muon-track i = 1 to M do
  (3) for each voxel j = 1 to N do
   (4) Compute C_{ij}, E-step
 (5) for each voxel j = 1 to N do
  (6) Compute $\lambda_{j,new}$, M-step
 (7) $\lambda_{j,old} = \lambda_{j,new}$
(8) return vector λ

4.3.4 EM Analysis

Whereas POCA was purely a heuristic algorithm that ignores the underlying physics of muons interaction, the EM algorithm is based on the theory of multiple scattering. Thus, while POCA performs well in simple situations where not much far spaced scattering occurs, EM should be able to better handle situations where multiple scattering is more frequent. This should result in EM being able to

discriminate between materials in realistic scenarios better than POCA can.

However, unlike POCA, EM is not a fast algorithm. The running time of the average EM algorithm is based on the number of muon events (M), the number of voxels (N), and the number of iterations run (I). Thus, the algorithm runs in $O(IMN)$ time. Information for every muon and every voxel also needs to be stored, so the memory usage is $O(MN)$. For the median EM, the performance degrades even more. Whereas the average method does not need to store all the correction values (C) for a voxel, the median method does. To find the median the values then need to be sorted. Assuming an optimal sorting routine is used this would result in a running time of $O(IMN[C\log(C)])$ and memory usage of $O(MNC)$.

The running times of the EM methods will be explored further in chapter 6. However, various ways to increase the running time of the algorithms were made in the implementation, both for the average and median methods. In fact, the original implementation of the median method resulted in running times so poor (over 24 hours) and memory usage so high (over 30 gigabytes depending on the scenario), the development of a faster method was necessary. These improvements and others will be discussed in section 4.3.5.

4.3.5 Improvements

The EM algorithm has advantages over other reconstruction techniques, but these come at a price. The EM algorithm has a slow run time in comparison to POCA and other reconstruction algorithms. In the course of research though, ways to alleviate these issues have been found. Some of these were implementation specific and will be explored in chapter 5, but a major change to the median method will be described here.

Using the median of the correction values as opposed to the average posed a hurdle because of the memory needed for storing the values as well as the extra computation time needed to sort them. Using the median method in this form was unfeasible. Instead an approximate median approach was developed. The aim was to find a way to lessen the impact of the non-Gaussian scattering (which the median method was supposed to do), without having to store all the correction values and needing to sort them. Techniques for finding the approximate median do exist but no guarantee is given to the quality of estimate. Since the EM is very sensitive to even small fluctuations in its internal calculations, more stability was needed from the approximation method. This led to the idea of using a binning method. The general concept of binning is to keep track of sums of similar values (eliminates the need for storage of all values) and then take the average of the group that would contain the median. It's illustrated graphically in figure 4.5.

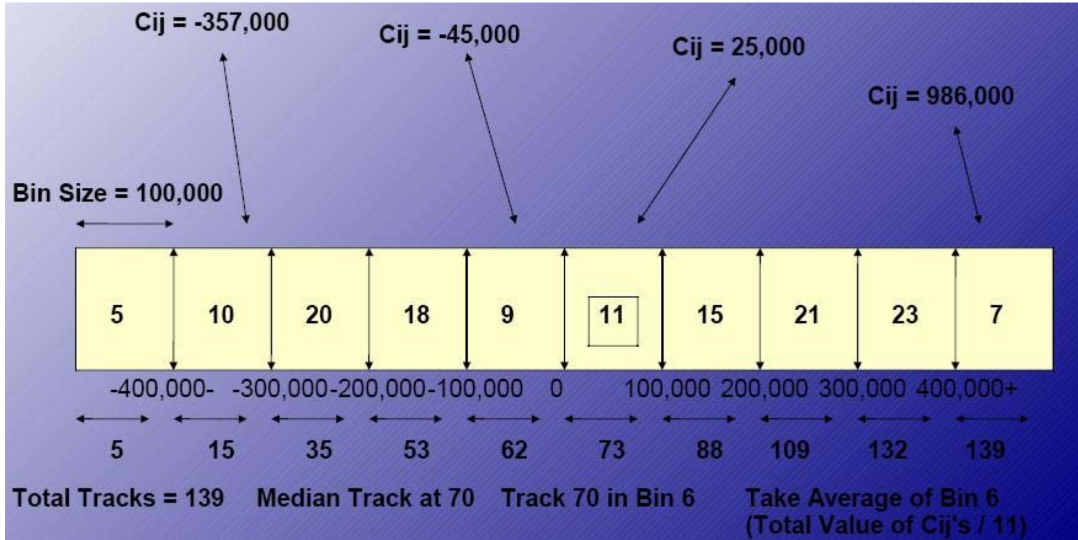


Figure 4.5: Using binning to find the approximate median

The number of bins and the size of the bins are set before the algorithm begins running. The bins are symmetric for negative and positive values, so if there were 100 bins the first 50 would be for negative values and the last 50 for positive ones.

The first and last bins are reserved for all values that exceed the minimum or maximum values that can be stored based on the number of bins and bin size, thus if there were 50 positive bins and the bin size was 10, any value over 490 would be stored in the last bin. When a correction value is found it is added to the appropriate bin based on its size and the size of the bin. For example, if there are 100 bins with a bin size of 10 and the correction value computed was 37, it would be added to bin 54 (the fourth positive bin in this case stores values between 30 and 40). The number of values in a particular bin also needs to be kept track of. Once all the correction values are processed, the bin containing the median (the total number of correction values for a voxel is already known, enabling it to be known which element would be the median; see section 4.3.2) is found and the total value in that bin is divided by the number of correction values that were added to the bin. For example, if there were three bins with 11 values each, the median would be the 17th value as there are 33 total values which has a median of 17. The 17th value would be in the second bin as the first bin has 11 values and the second bin has 11 for a total of 22 values in the first two bins. The average value of the second bin is then approximated as the median. This number is used to update lambda instead of the average or true median correction value.

To get a good estimation of the median the number of bins and their size is of high importance. The distributions of correction values of many voxels were studied to determine what these parameters should be. Fortunately the absolute values in the distributions were very similar and evenly divided in the first several iterations for most voxels. This meant that having a static number of bins and bin size would not cause uneven binning depending on the voxel. Eventually as the lambda values for the voxels converge, the correction values decrease. As they decrease they tend to be binned together more because the bin size is static throughout the entire run. Thus the approximate median shifts from being closer to the true median to being

closer to the average. However, since this happens once the algorithm approaches convergence, the average and median values are relatively close and this does not pose as large a problem as it would if it happened in the earlier iterations. After this study was done 200 was decided on as the number of bins to use and 100000 as the bin size for C_{ij}.

Chapter 6 details experiments done to compare the run times of the different methods, but the approximate median algorithm should run faster and make more efficient use of memory based on complexity analysis. Since the number of bins (B) is constant, no matter how many rays pass through a voxel the storage needed remains the same. This means memory consumption is $O(MNB)$ as opposed to $O(MNC)$. This doesn't necessarily mean the approximate median will always have the advantage over the true median as the number of correction values could be less than the number of bins. In the course of experiments though, this has rarely been the case as the number of bins has been at most 200, and in simulations the typical number of correction values is in the thousands. Besides the savings in memory the biggest improvement of the approximate median over the true median is the decrease in the running time. The approximate median requires no sorting and the worst case scenario would be that when finding the median it's stored in the last bin, as all the bins would have to be processed. This makes the runtime $O(IMNB)$, which for the experiments of this study has been much better than the runtime of the true median which was shown to be $O(IMN[Clog(C)])$.

The approximate median was an important development in the course of research into the EM algorithm and was tested thoroughly. The results obtained from this new method will be discussed extensively in chapter 6 and compared against results from POCA and the average EM. The run times of the different EM methods will also be explored.

Chapter 5

Implementation and Methodology

5.1 Simulation Overview

All the results from the algorithms that will be shown are run from data created in simulations. The process from simulating data to analyzing results has been fine tuned over the course of the research. Currently the general course is to first create a simulation in Geant4. Next, compile and run the program and output the data to a file. Finally the reconstruction algorithms are run on the data which write the results to another file, to then be analyzed with the appropriate tools.

5.2 Tools

Besides the reconstruction algorithms themselves, a variety of tools are used for simulation and analysis. Geant4 and CRY are the software packages used to produce the input data for the reconstruction algorithms to be run on. ROOT is then used to plot and analyze the output from the algorithms. The simulation tools will be described first.

5.2.1 Geant4

The Geant4 software package is used to simulate the passage of particles through matter [43]. All aspects of detector simulation are covered as it contains the following tools: geometry, tracking including multiple scattering, detector response, run, event and track management, visualization and user interface [44]. To handle the multitude of different fields and applications Geant4 can be used for, it models a massive set of physics processes that oversee a vast array of interactions of particles with matter over an extensive energy spectrum. It is written in C++ and has an object-oriented design which facilitates easier understanding, extension, and

customization of the toolkit. It was developed by RD44, a collaboration of over 100 scientists from all over the world [45]. Geant4's physics have been validated multiple times and the tests it has gone through are detailed thoroughly in “Geant4 Developments and Applications,” by J. Allison, et al. [43].

For the purpose of this study, Geant4 was used to create simulations modeling the passage of muons through matter. This involves creating the geometry of the scenario, such as the detectors and objects inside, as well as modeling the physics processes relevant to muons (i.e. the Coulomb force). Much work is done writing the code for the scenarios. Besides the geometry itself, a driver and stepping action for the program need to be created as well which are used to guide the program, and track particles and output data. Once the program is complete, it is compiled and run. The program then uses Monte Carlo methods and advanced random number generators [44] to simulate the passage of the muon through the user created geometry.

In the stepping action the muon particle can be tracked discretely. This is where the events are output to file. In the current set up, only events that hit both sets of detectors in a scenario are printed to file. Generally the scenarios run have three top and bottom detectors as seen in figure 5.1

The point where the muon hits each detector is output to file sequentially. Also, before the points of the top detector are output, the momentums of the muon at the top and bottom of the volume is printed out. This is all the data needed to begin running the reconstruction algorithms.

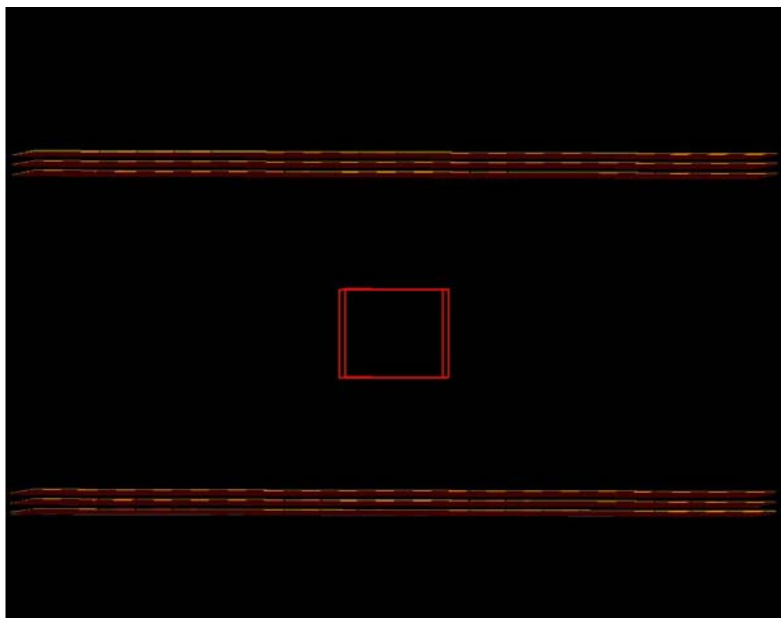


Figure 5.1: Simple Geant4 scenario: three detector planes above and below an uranium block

One thing Geant4 doesn't provide is a built in package that models cosmic ray muon energy and angle distributions. Another software package that was used to do this is described in the next section.

5.2.2 Cosmic-Ray Shower Generator (CRY)

CRY is a software library that generates cosmic-ray particle shower distributions at different elevations for use as input to software that simulates the passage of particles through matter, like Geant4. It was designed primarily for use in transport and detector simulation code [46]. It uses Monte Carlo methods to model the Earth's atmosphere and to produce the corresponding cosmic ray showers [10]. It is callable from C++ and FORTRAN and interfaces smoothly with Geant4 [46]. It has been shown to produce distributions that match up quite well with results obtained experimentally [10][46]. Figure 5.2 shows how the energy distribution of muons at sea level produced by CRY compares with the experimental results found by B.C. Rastin [51]:

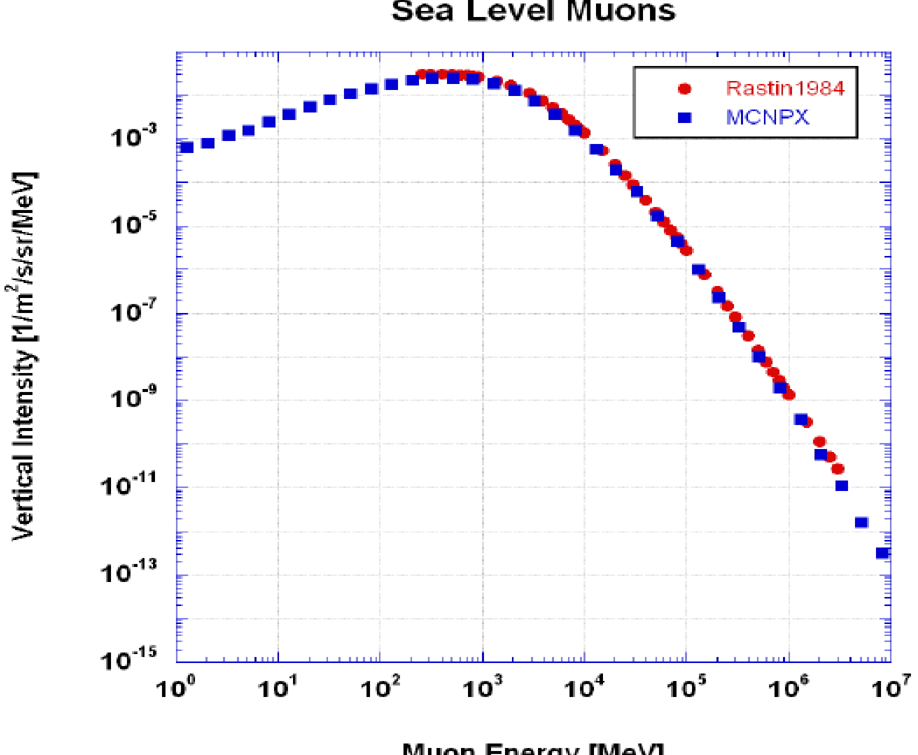


Figure 5.2: CRY generated muon spectrum at sea level against experimentally determined spectrum [46]

For our purposes, CRY was interfaced with Geant4 so that when the scenarios were run they used the true distribution of cosmic ray muons with respect to the energies and incoming angles at sea level. This provided the reconstruction algorithms to be tested on more real life data than Geant4 could have provided alone.

5.3.3 ROOT

ROOT is a data analysis tool developed at the European Organization for Nuclear Research (also known as CERN) [47]. The ROOT system provides a set of object oriented frameworks that offer the functionality needed to process and analyze huge amounts of data as efficiently as possible. ROOT has a built in CINT C++ interpreter so the command, scripting, and programming language are all

C++ and the tedious compile and link paradigm is avoided. Some of the features included in ROOT are histogramming methods in any number of dimensions, curve fitting, function evaluation, minimization, graphics and visualization classes that allow easy querying and processing of the data interactively or in batch mode [47].

ROOT is the analysis tool used most by this study. Scripts were written that processed the data produced by the reconstruction algorithms and displayed it in a variety of ways. Chapter 7 shows the results from the reconstruction algorithms and there the full power of ROOT will be seen.

5.3 Muon Tomography Suite

The muon tomography suite (MTS) was built as a standalone application written in C that would run the POCA and EM reconstruction algorithms. It had one major revision as well as countless updates and improvements, but was developed modularly and using stepwise refinement. Apart from producing reconstruction output, the suite has evolved and can be used for many other processing needs as well. The general working of the program will be described in this section as well as interesting implementation details.

5.3.1 Driver Implementation

The MTS was going to be an expansive program with many options on how to run. Thus it was developed as modularly as possible. A driver module was created that handled all input, parameter setting, and file manipulation. MTS may be run by using command line options or through use of a configuration file provided as the only argument to the program. Getopts, a very useful C tool for Unix systems, was used to provide the great array of options available to the program while keeping the code from getting unwieldy and maintaining it to be easily understandable. The main parameters needed to run are the input file and the reconstruction or analysis

to be done, but the following options were also able to be set on the command line or configuration file (otherwise default options would be used):

4. input file name
5. reconstruction algorithm to be run (EM median/average or POCA)
6. various EM parameters (voxel size, bin size, number of bins, iterations, nominal momentum)
7. optional output (non-reconstruction information like data distributions, internal calculations to the algorithms, and debugging information)
8. the metric units to run the algorithms with
9. cuts to use in the algorithm (such as momentum and angle cuts)

Other options were included or taken out as well, but the ones listed were the main options that are still included in the program and are useful. All parameters were stored in a single array. The pointer to the array was passed to all other modules of the program so that access to all options and settings were available everywhere in a consistent and succinct way.

The driver module also handled all file manipulation. All needed files were opened in this module, whose pointers were stored in an array and passed to all modules needing file access. The same was done for output files. All standard output file names were created by the driver module. They contained the same pre-extension name as the input file, but different extensions were added depending on the options run with the program to enable easier organization and the ability to recognize what type of reconstruction was done. All optional output had the file names included as an option or were given default file names. Once all the parameter setting and file manipulation is done, the reconstruction algorithm to run is called. POCA and EM can be run independently for reconstruction, but the

POCA module is run for both algorithms as the input for EM is based on POCA results. The next sections will show how they were implemented.

5.3.2 POCA Implementation

The implementation of the POCA algorithm is quite straight forward. The algorithm is simple geometry and translates well to code. A separate library of vector functions was created for use in the operations needed in POCA and other parts of the MTS, like dot product and angle calculations. Besides this, the POCA algorithm module is very simple. However, when the POCA module is called by the driver it's much larger and in depth than just the POCA algorithm. It does a variety of analysis as well as preparation for EM.

After initializing all necessary data structures, a header function is called that reads the first several lines of the input file. This gets information such as reconstruction volume size, as well as number of events. It then sets parameters based on this information such as the number of voxels and the range of distances in the different coordinate directions. Next, a while loop is entered that runs until all data from the Geant4 input file is read. Each event is handled independently. The six points where the muon hits the top and bottom detectors are read in as well as the momentum for the muon. Next, a least mean squares algorithm is run that fits the points from the top and bottom detectors into two vectors. Finally the POCA algorithm is called on the two vectors and the scattering point is determined. The POCA algorithm module returns the scattering points and the distance of closest approach (the length of the line segment between the points of closest approach) or some error value, such as lines being parallel, that will tell the program whether to continue with analysis of this event or not. If the lines are parallel, or there was some other error, then all other analysis is ignored and the next event is processed, otherwise the scattering angle is determined using the following equation:

$$\theta = \cos^{-1}\left(\frac{a \bullet b}{|a||b|}\right)$$

where **a** and **b** are the vectors of the incoming and outgoing muon rays.

After the scatter point and scattering angle is found, more work is left to do depending on the options selected including formatted output written to file. The other main purpose of the POCA module is to do the preprocessing work needed for EM, such as collecting the input and estimating the muon path in the volume and collecting the needed information in regards to that. This preprocessing for EM will be covered in the following sections, but first the data structure created for EM will be analyzed.

5.3.4 EM Data Structure

The EM algorithm needs a substantial amount of data while running. Information for every voxel that a muon passes through needs to be stored. This poses a significant memory issue as scenario's may be run where there are millions of muons and hundreds of thousands of voxels. The structure for EM was developed to be as memory efficient as possible as well as to provide easy implementation of the EM algorithm, while allowing the ability to fluidly add functionality to the program. It is a C struct and is displayed in the next figure.

This structure contains all the information about a particular muon. All muon events are independent of each other, so they can be accessed sequentially and no random access to a particular muon is needed, thus a linked list structure was used to hold all the different muon events.

```
struct muon {
   unsigned int event;
   double dtX, dtY, dX, dY, pr2;
   double sigma[3], sigmaMed[3];
   struct voxel* muonTrack;
   struct muon* nextMuon;
};
```

Figure 5.3: Main structure used to encapsulate data for the EM algorithm. The structure contains the information regarding a particular muon.

The first line is the event variable that was used for debugging purposes. This number remains the same through the data handling in the POCA module until the EM algorithm finishes, so it allows easy event tracking which proved quite useful as will be highlighted in chapter 6. The input information, $\Delta\theta_x$, $\Delta\theta_y$, Δ_x, Δ_y, and p_r^2, are stored respectively in the variables on the second line. The third line contains two arrays that store the covariance matrix. There is one each for the average method (sigma) and the approximate median method (sigmaMed) so that both can be run simultaneously. This is one area where a moderate amount of memory was saved compared to the straight forward implementation of the EM algorithm. The covariance matrix has four elements but the second and third elements always contain the same element, so having four elements in the array is superfluous. In addition to storing the covariance matrix, the arrays are also used to store the parameter V_{ij} (the inverse of the covariance matrix) as the original covariance is not needed after its' inverse is found. This saves a sizable amount of memory. Doubles on our 64-bit systems are 8 bytes, so 80 total bytes per muons are saved . The largest scenario we attempted to run contained 10 million muon events so this one change can save upwards of 800 megabytes for large runs. The next line

is another important part where much memory was saved relative to the normal implementation of the EM algorithm. It is a pointer to another structure that contains the information needed for each voxel. It's illustrated in figure 5.4.

```
struct voxel {
   unsigned int ID;
   double wt, wtX, wX;
   struct voxel* nextVoxel;
};
```

Figure 5.4: Sub-structure used for EM algorithm data. This structure contains all information needed for a voxel that a muon passes through.

The EM algorithm was shown to require estimates of a muon's path through the reconstruction volume. From this path two parameters are gleaned. *L*, which is the path length of a muon through the current voxel, and *T*, which is the length of the muon paths from after it exits the current voxel until it exits the reconstruction volume. However, these parameters are static throughout the entire algorithm and are only needed to determine the weights (used for the calculation of the covariance matrix and are also static) for the voxel. This makes storage of the L and T parameters unneeded after the weights for the voxels are determined. Since the number of voxels a muon can possibly travel through can be quite large (hundreds of thousands), and the number of muon events high as well (millions), this provides a substantial memory saving compared to the normal implementation of the EM algorithm. The voxel structure also is a linked list that contains all voxels a particular muon hit, and the last line in the structure is the pointer to the next voxel.

Now that the data structure used for this implementation of the EM algorithm is

detailed, the EM algorithm implementation itself can be explained. First the computation of the input data for the algorithm will be displayed.

5.3.4 EM Preprocessing Implementation

The EM algorithm requires that a sizable amount of data be processed before the iterative phase begins. This information is based upon the estimated track of the muon through the volume it's passing. The path used in this implementation is determined by finding the scattering point of the muon and connection the incoming muon track to this point, and then connecting the point to the exiting muon track. If there is no scatter point (i.e the tracks are parallel) then the straight line projection is used by connecting the incoming muon track to the outgoing track. This process takes place in the POCA module as detailed in section 5.3.2. Once this is done, the new instance of the muon data structure is created and the first part of the preprocessing for EM is done.

The EM algorithm requires the displacement and scattering angle in the x and y directions for a particular muon, as well as the momentum parameter, for input. The process of determining these was explained in chapter 4. The implementation was done in the exact manner laid out. Once the parameters are determined they are stored in the muon data structure. This structure is then passed to another module for determining the voxels a muon passes through on its course through the volume.

Tracking a muon's actual path through the volume is not possible, but a good prediction can be made of what voxels the muon traveled through. This is necessary for the EM as well as determining a muon's path length through a voxel and distance left it has to travel through the volume (parameters L and T). This is handled by creating a line from the incoming muon track to the scatter point (or the

incoming track to the outgoing track if there was no scatter point) and a line from the scatter point to the outgoing track. These lines are defined parametrically. The muons initial position in the volume is determined and then what voxel it is in can be calculated by an arbitrary predetermined numbering scheme. The boundary coordinates of the adjacent voxels are then computed, and the time parameter (t) is calculated for the parametric equations for when the track is at boundary coordinate. The voxel with the minimum time parameter is chosen as the next voxel that is reached. The coordinates of the current voxel are then set to the entry point point of the new voxel and the process is repeated. This is done until the end of the volume is reached.

When the next voxel to be traveled to is found, L can be determined by using the distance formula with the initial point the muon entered the current voxel and the point where it exits the voxel. The same concept can be use to find T, except using the point where the muon exits the current voxel and the point where it exits the volume. As was explained in the section on the EM data structure, these parameters are not stored and just used to determine the weights for the voxels and then discarded.

Once the input data is processed, the muon track estimated, and the weights for the voxels ascertained, the preprocessing is finished and the EM algorithm may begin.

5.3.5 EM Implementation

The EM algorithm is relatively straightforward to implement, after the input data is determined. After all data structures are initialized and the input data passed in, the main "for" loop is entered where the expectation and maximization steps are iteratively run. Here computation time is saved by altering how the algorithm calculates the correction values for the expectation step. The inverse of the

covariance matrix is needed to get the correction value for a particular voxel (see section 4.3.2) and the algorithm calls for getting the inverse covariance matrix for all muons first, then going through all muons again to calculate the correction value for all the voxels they went through. All correction values are based on the information provided by a single muon and all muon events are independent, so redundant work can be avoided by determining the inverse covariance matrix for a muon and then immediately determining correction values for all voxels that muon goes through. These are either added to a variable storing a running tally of correction values for a voxel in the average method, or are binned for a voxel if the approximate median method is being run. This change prevents processing all muons twice, which is significant as millions of muon events are typically used in reconstructions.

After the correction values are processed, the average or approximate median is found described by the methods in the chapter 4, and the lambda value for each voxel is updated. Once all iterations are complete, the final values of lambda along with other voxel information is output to file and the suite takes care of freeing data structures and the program ends.

5.4 Software Testing

The reconstruction process requires the sharing of information between many different programs and modules. This unfortunately results in the process having many areas for failure. Due to this, software testing was a large part of MTS development and ended up illuminating better ways to structure the suite as well as fixing bugs that produced inaccurate results. The software techniques used to analyze the MTS are explored in this section.

5.4.1 General Techniques

Testing was implemented throughout the software development cycle. General techniques included both black box and white box methods. Unit testing was used extensively to validate the output of modules after they were developed. This was used with good results on the vector functions that were implemented as well as on internal routines needed for the reconstruction algorithms such as the POCA algorithm module, the preprocessor for the EM input, and the modules that produced the correction values for the EM algorithm. The output of the individual modules was produced on known input and compared to what the output should be (calculated by hand). The same process was used with boundary values on the modules where it was applicable (for example the module that estimates the muon tracks) to ensure that they behaved correctly on extreme cases. This enabled validation of the modules as well as uncovering bugs that were fixed before the entire reconstruction implementation was even complete.

A white box technique that was heavily used in the testing of the EM algorithm was the analysis of the convergence of the lambda values of voxels inside and outside the targets in a simulation. The lambda value of the voxel being analyzed was output at certain iterations. Voxels that displayed interesting behavior (such as not converging, or converging to an unlikely value) were then looked at more in depth. This resulted in the discovery of several small bugs and implementation issues that were fixed. Similarly, the correction values of particular voxels were also explored which resulted in several interesting findings (a specific example will be given in section 5.4.2).

Static techniques were also used, and this included code review. Dr. Debasis Mitra regularly reviewed the MTS code to ensure proper implementation of the algorithm in addition to checking for mistakes. This resulted in high assurance that the

algorithms were implemented correctly.

Besides testing to ensure accurate results, the stability of the program was also examined. The program in general was coded defensively to make sure crashes and unpredictable behavior was kept to a minimum. This was an important issue because the EM algorithm has a relatively long run time and regular crashes of the program would have seriously hampered research as certain simulations took many hours to reconstruct. To test the stability of the MTS, fuzz testing was used on the modules that handled input and output. This involved giving random or incorrect input to see how the program handled it. Mutation testing was also used to alter internal data and see how the program responded. Through these kinds of testing several faults were found in the I/O and other routines and were fixed. This resulted in few unexpected program terminations and the MTS running stably.

The techniques described in this section were mostly used from the beginning of development to help minimize errors in the code and prevent more complex testing and difficult debugging later on in the cycle. Some other interesting techniques were used for specific larger problems and will be detailed in the next section.

5.4.2 Specific Examples of Software Testing

A successful black box method used for debugging the MTS code was comparing the results of two different implementations of the same algorithms. Dr. Kondo Gnanvo also used the POCA algorithm to reconstruct Geant4 simulations [17]. His output for POCA was compared to the output from the POCA in the MTS and it was discovered that the scattering angle distribution for the MTS was incorrect, which caused problems in the results for POCA and for EM as they both use this information. Once this was determined the module for angle calculation was found to have improperly handled the vectors representing the muon tracks. This was

fixed and the distributions compared again, revealing a proper distribution. A ROOT plot showing the distributions from Dr. Gnanvo's code and the MTS code is found in figure 5.5.

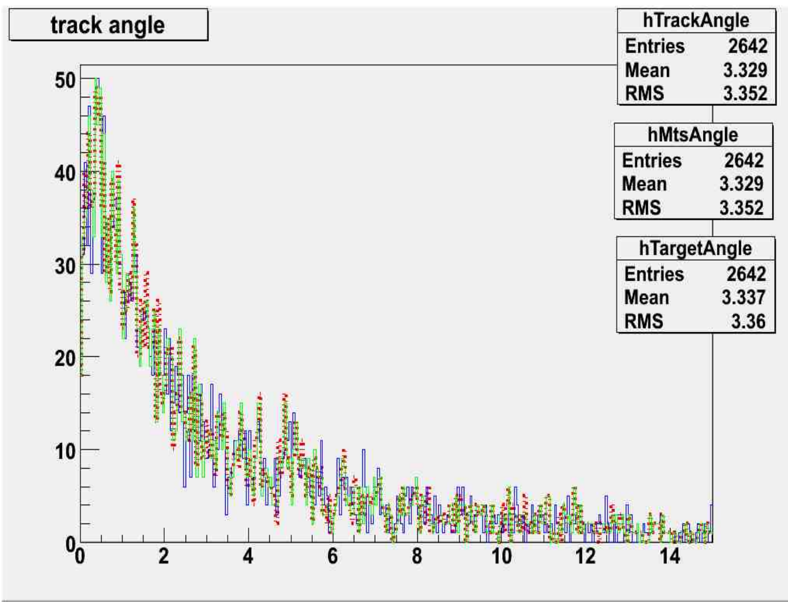


Figure 5.5: Scattering angle distributions from Dr. Gnanvo's POCA and the MTS POCA. The x-axis represents the magnitude of the scattering angle while the y-axis represents the number of scattering angles.

After fixing the problem with the scattering angles, the reconstructions produced were much better than the ones produced before in terms of discrimination of targets and noise reduction.

A source of contention throughout the research of this study was whether or not certain inconsistent behavior was due to implementation errors or faulty input from Geant4. The output directly from Geant4 was verified to be correct many times throughout the course of this study. However, there were still found certain strange results that couldn't be explained by a fault in the MTS. One of these issues was that incoming and outgoing tracks that were in essence parallel, were still being

used to find POCA points (review section 4.2 to see why this shouldn't happen). These tracks were unscattered and thus should have had no scatter point. Also since scatter points were found, the scattering angle and displacement were found as well. This led to poor results in both POCA and EM reconstructions. As mentioned, Geant4 output had been verified as correct in the past, so this led to the belief that there was a problem with the least mean squares fit of the tracks. This function had been independently tested by the techniques described in the previous section, so it would have been surprising if it had a problem with so many 'normal' tracks. A test idea was then devised.

The Geant4 simulations were run with ideal conditions, so that no scattering occurs in the GEM detectors and all detector points are on the same line. If the parallel tracks were used directly from Geant4 and not fit, they should produce the correct behavior in terms of parallel track detection. Once this test was run it was determined that the parallel tracks were still not detected properly, proving there was an issue with the input data from Geant4. Analysis of the data from Geant4 revealed that the precision of the coordinates printed to file was not perfect and the values were being truncated. This meant that even though the tracks truly were parallel, the level of precision prevented the MTS POCA from detecting them. After this was found the precision of output was increased and the issue of parallel tracks with scatter points was resolved.

One problem that was fixed used a mix of both black and white box testing. Based on reconstruction plots from the EM algorithm run on many different simulations it was seen that some voxels were getting high values of lambda yet did not contain any material and were unlikely to contain tracks that were scattered at all. This was an odd observation and led to analyzing those voxels specifically and the correction values they contained. After pouring through the correction values it was found that

a few events had extremely high values that were many magnitudes larger than the majority of values. More in-depth analysis revealed that these tracks had displacements in the X direction of several meters which would be nearly impossible if no scattering occurred and highly unlikely even if some small scattering did. This led a review of the module handling the computation of the displacement values. It was determined that these large displacement values occurred when the tracks were completely perpendicular to the Y-axis. This observation led to the idea that the least mean squares fitting of the muon tracks had a bug. Careful review of this routine led to the discovery of an incorrect condition check for when tracks were perpendicular to the Y-axis. Once this was rectified, the large displacements disappeared and along with them all the problem voxels.

Chapter 6
Experiments and Results

6.1 Introduction

The following are results produced from ROOT visualization based on the data provided by the MTS program. They are divided into results from POCA reconstruction, EM reconstruction with the average method and EM reconstruction with the approximate median method. Preceding the results is a brief brief section describing the scenarios to be modeled. The results are followed by a section further analyzing them and a final section that analyzes the algorithms' computational performances.

6.2 Scenarios

The next several sections will define the scenarios that were simulated with Geant4 to run the reconstruction algorithms on. The specifics of the scenarios will be described in the appropriate sections, but they all share some similarities detailed here. All of the scenarios are run with ideal conditions. They are run with vacuum as the background; the only material being the target objects inside the volume. The detectors are modeled in such a way that they provide no scattering. In addition, the detectors operate at perfect spatial resolution. Every simulation was run with the equivalent of 10 minutes exposure time. The reconstruction parameters used for the EM algorithms are also the same. All scenarios use 5cm X 5cm X 5cm sized voxels except for the vertical clutter scenario which uses 10cm X 10cm X 10cm voxels. The number of iterations run is 200. One feature that not all the scenarios share is reconstruction volume size. This information will be provided with the description of each simulation.

For all plots shown in this chapter, the Z-axis is oriented vertically, with the Y-axis being on the left side of the plot and the X-axis on the right. The unit of length used in the plots are millimeters.

6.2.1 Basic Scenario

The basic scenarios are all of the same geometry, namely a 10cm X 10cm X 10cm box in a volume sized 200cm X 200cm X 110cm as seen in figure 6.1 produced directly from Geant4 simulation codes:

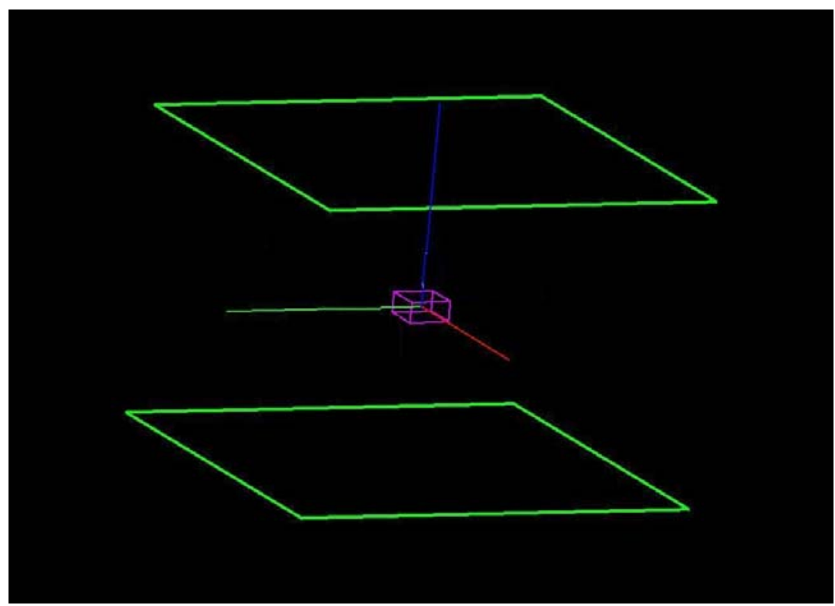


Figure 6.1: Prototype of the basic scenarios: A single 10cm X 10cm X 10cm box of varying materials in a 200cm X 200cm X 110cm volume

Five different materials were modeled, aluminum, iron, lead, tungsten and uranium. These were chosen as aluminum and iron are low and medium Z type materials, respectively, that would most likely be encountered in the types of cargo being imaged. uranium was obviously chosen to represent a threat object while lead and tungsten were used to represent high Z materials that might be found in real world

scenarios that come closest to having the same properties uranium has in regards to muon scattering.

For the EM algorithm these scenarios will be used as a baseline for other scenarios. Accuracy analysis is done to find a threshold for lambda where the reconstruction comes out best in terms of the properly reconstructed voxel versus false positives and false negatives. For example, if a threshold of 10,000 is chosen for uranium, all voxels inside the target that are above this value will be counted as true positives and those below will be considered false negatives. Any voxels not in the target that are above this value will be categorized as false positives. Once the thresholds are determined for each material in the basic scenarios, this analysis can be used in the more complex scenarios to see how well the algorithm is discriminating between different materials, which is its most important goal.

6.2.2 Five Target Scenario

The five target scenario consists of five 10cm X 10cm X 10cm boxes in a 200cm X 200cm X 110cm volume placed symmetrically on the same z-plane, using the same materials found in the basic scenarios. aluminum is centered at -500cm x, -500cm y, 0cm z, iron at -500cm x, 500cm y, 0cm z, lead at 0cm x, 0cm y, 0cm z, tungsten at 500cm x, -500 y, 0cm z, uranium at 500cm x, 500cm y, 0cm z. The Geant4 geometry is displayed in figure 6.2.

The thresholds for the materials that were found using the basic scenarios will be used first in this scenario to see how well discrimination can be done in a relatively simplistic situation.

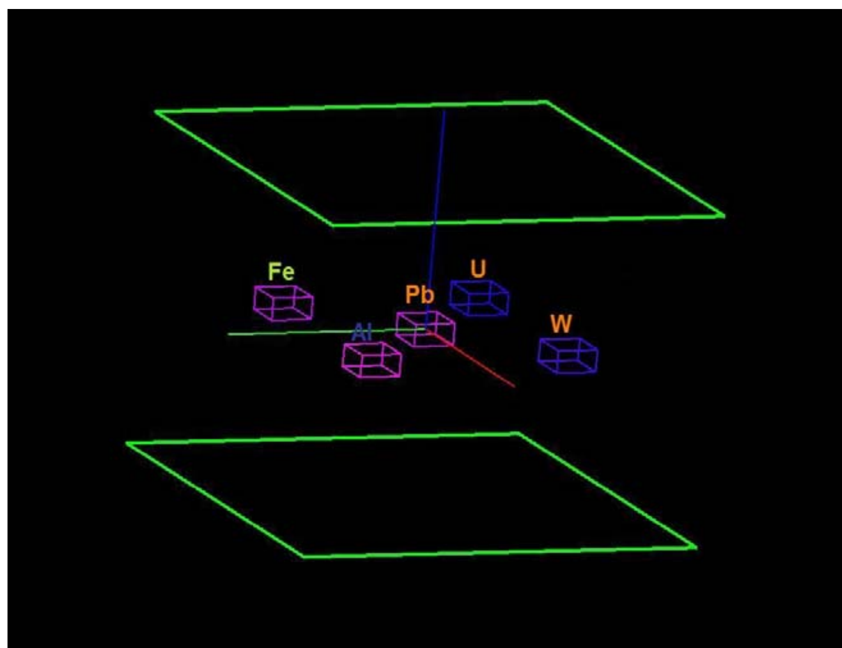


Figure 6.2: Geometry of the five target scenario: Five 10cm X 10cm X 10cm boxes of varying materials (Al, Fe, Pb, W, U) in a 200cm X 200cm X 110cm volume

6.2.3 LANL Scenario

This simulation is an attempt at recreating a scenario used by the team at Los Alamos National Laboratory in their paper introducing the EM algorithm [19]. The 200cm X 200cm X 110cm volume contains three 10cm X 10cm X 10cm boxes. The materials used are tungsten (-300cm, -300cm, 300cm), iron (0cm, 0cm, 0cm), and aluminum (300cm, 300cm, -300cm). This setup is displayed in figure 6.3.

This scenario will be used to generally compare how this studies implementation of the EM algorithms compares to the original. Accuracy analysis will also be done with the thresholds calculated in the first scenario.

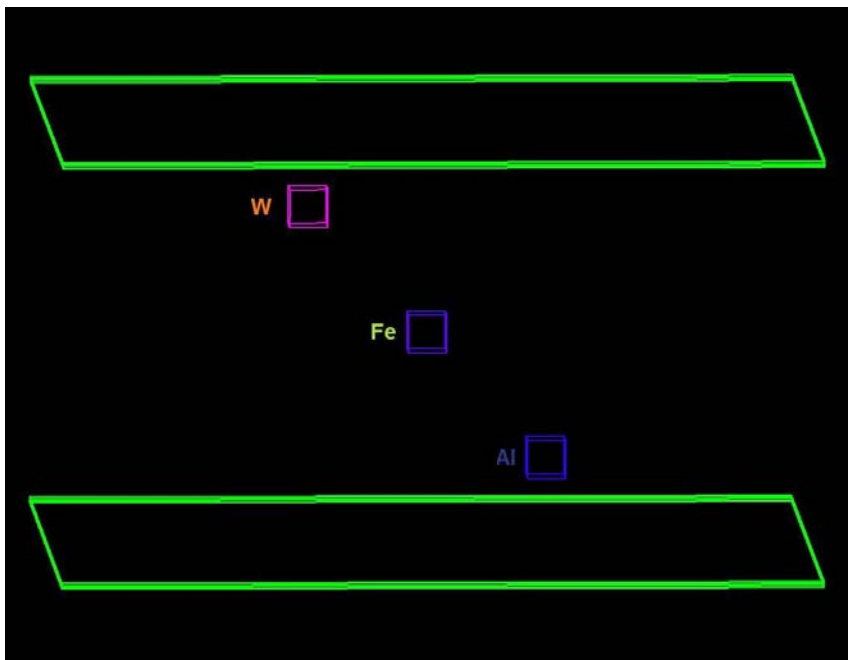


Figure 6.3: Geometry of the LANL scenario: Three 10cm X 10cm X 10cm boxes of varying materials (W, Fe, Al) in a 200cm X 200cm X 110cm volume

6.2.4 Vertical Clutter Scenario

The vertical clutter scenario is significantly different from the preceding simulations. The reconstruction volume is significantly larger at 400cm X 400cm X 300cm. It contains three 50cm X 50cm X 20cm rectangular boxes stacked vertically. An iron box is centered at 0cm x, 0cm y, 0cm z. Above the iron box is a tungsten one centered at 0cm x, 0cm y, 30cm z, while below is an aluminum box centered at 0cm x, 0cm y, -30cm z. Also, for the reconstruction, 10cm X 10cm X 10cm voxels were used due to larger targets objects being reconstructed. The geometry for this scenario is displayed in figure 6.4.

Scenarios with vertical clutter are a situation that POCA has been shown to have trouble with reconstructing well in comparison to how it performs with non-vertically oriented scenarios [11]. The EM algorithm is supposed to have less

trouble dealing with this type of setup; the analysis of this simulation and its comparison with POCA will show if this is the case.

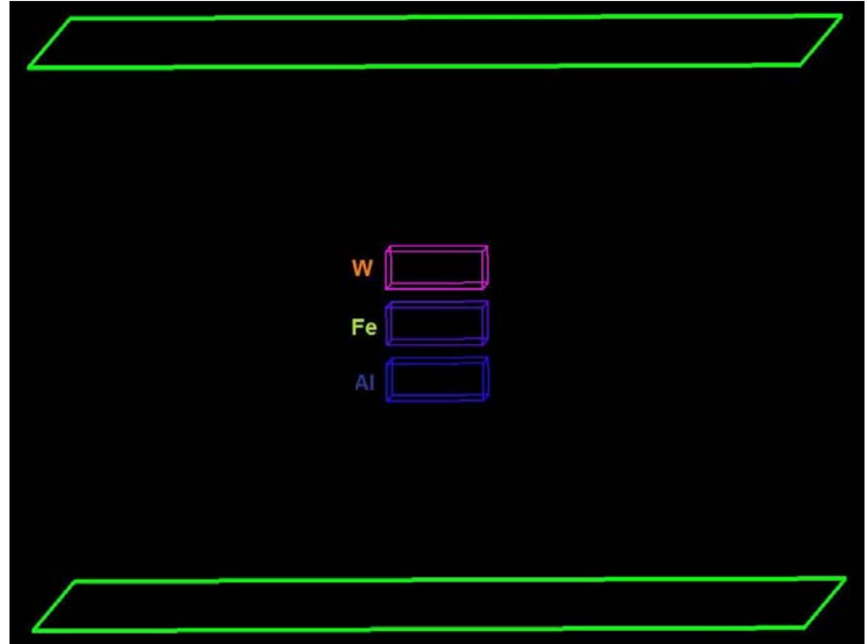


Figure 6.4: Geometry of the vertical clutter scenario: Three 50cm X 50cm X 20cm boxes of varying materials (W, Fe, Al) stacked vertically in a 400cm X 400cm X 300cm volume

6.2.5 Truck Scenario

This scenario is a simulation of potentially what muon tomography is being developed for. A truck containing various liter sized objects (10cm X 10cm X 10cm) placed at different positions was created using Geant4. The reconstruction volume is very large at 720cm X 360cm X 360cm. Features of the truck include the engine, frame and chassis consisting of iron, a battery made out of lead, a windshield containing glass, and tires of rubber with iron in the center modeling the rims. The liter sized objects in the truck are made up of 12 uranium blocks, 3 tungsten blocks, 3 lead blocks, 7 iron blocks, and 3 aluminum blocks. The geometry is displayed in figure 6.5:

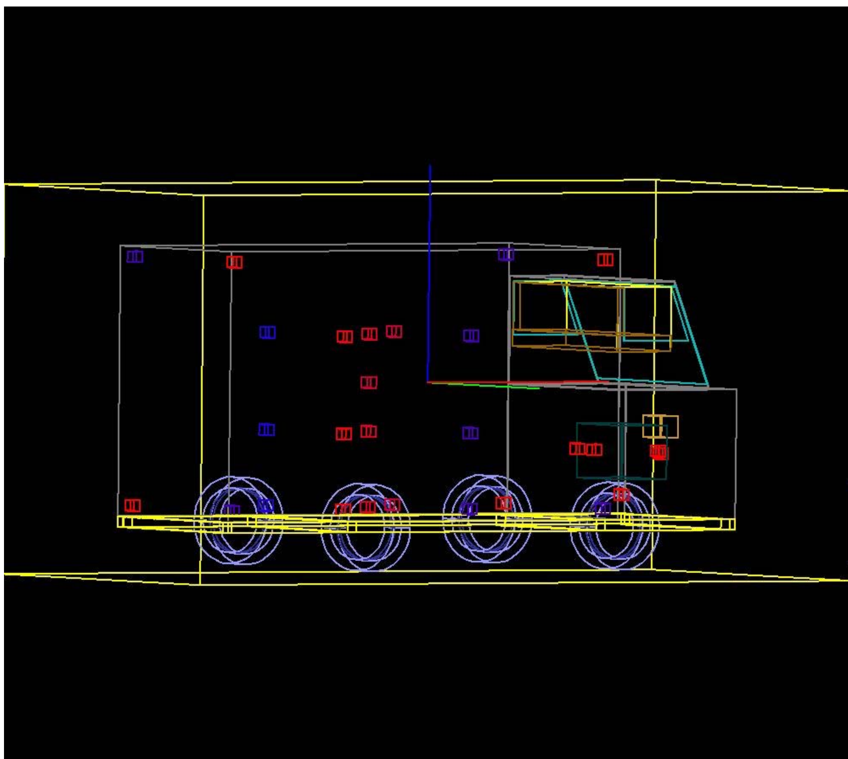


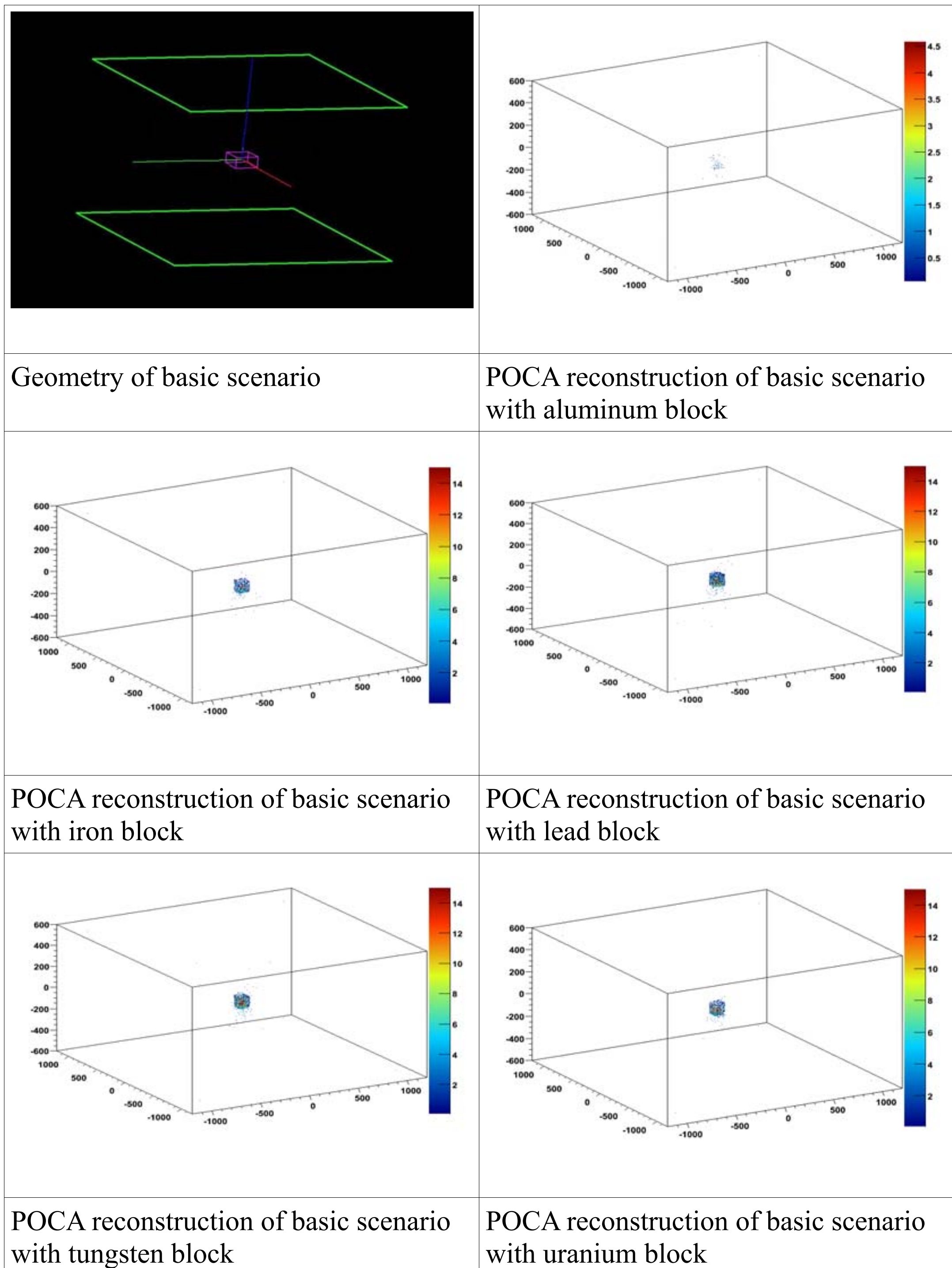
Figure 6.5: Geometry of the truck scenario: The blue and red boxes in the figure are 10cm X 10cm X 10cm. The red boxes are high-Z material like uranium and tungsten. The blue boxes are medium-Z material like iron.

This scenario is the most realistic of all the simulations. The main analysis of this situation will be to see how well EM can differentiate higher Z materials from medium and low Z materials, using the thresholds found in the basic scenarios.

6.3 POCA Results

This section consists of the results of the POCA reconstruction algorithm used on the simulations produced by Geant4 that were detailed in section 6.2. The color scale for the plots (seen on the right side of the figures) represents the magnitude of the scattering angle in degrees.

6.3.1 Basic Scenario



Geometry of basic scenario

POCA reconstruction of basic scenario with aluminum block

POCA reconstruction of basic scenario with iron block

POCA reconstruction of basic scenario with lead block

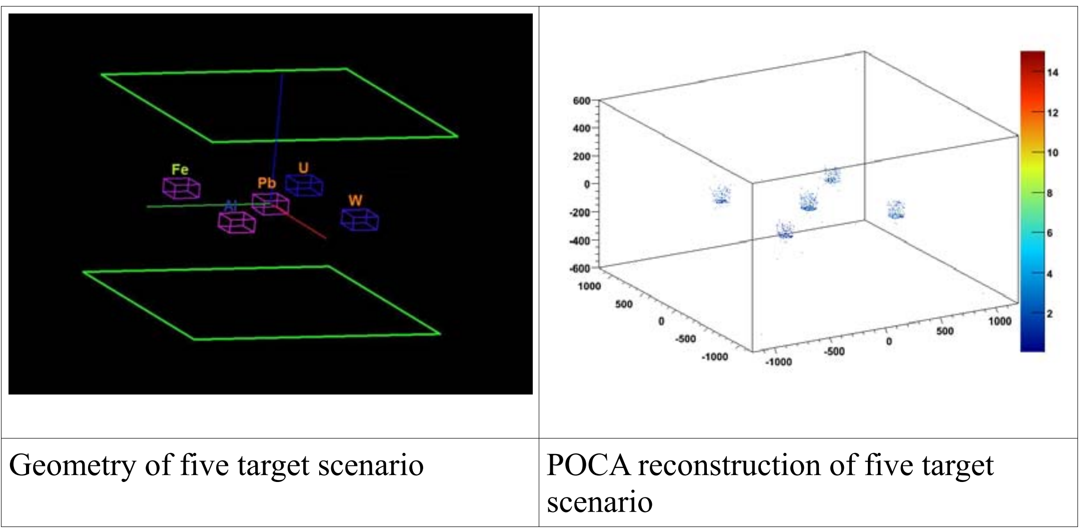
POCA reconstruction of basic scenario with tungsten block

POCA reconstruction of basic scenario with uranium block

Figure 6.6: POCA reconstruction of basic scenario with different materials

The POCA reconstructions for these simple scenarios come out quite well as seen in figure 6.6. The shape of the box is definitely made out in all the plots, though it is not quite as apparent in the aluminum reconstruction. This is due to the lesser scattering of the muons through the aluminum block and more tracks being calculated as being parallel and thus fewer muon tracks having scatter points. Also, the relative scattering for the materials is clear. The plots are normalized to each other and as the material changes between elements of higher atomic number, the number of higher scattering angles can be seen to increase. However determining what a material is based only on the 3D plot of its scatter points is a difficult proposition due to the nature of the POCA algorithm.

6.3.2 Five Target Scenario



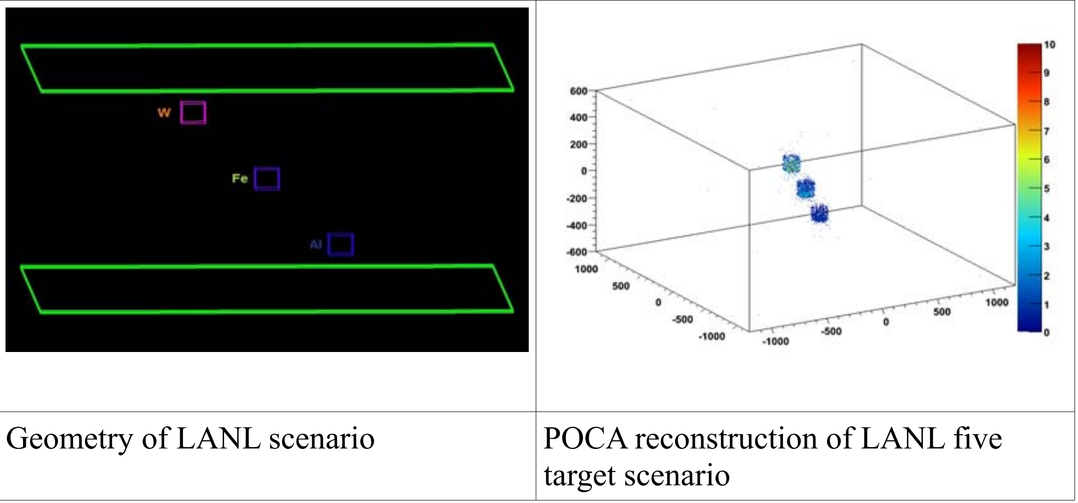
Geometry of five target scenario	POCA reconstruction of five target scenario

Figure 6.7: POCA reconstruction of five target scenario

The POCA reconstruction of the five target scenario (Table 6.2) shows similar results to the basic scenarios. The five targets are reconstructed in a box shape with the higher Z materials (uranium, tungsten and lead) displaying more scatter points with high scattering angles than the medium and low Z materials (iron and aluminum respectively). This again shows the success of POCA at constructing

simple scenarios.

6.3.3 LANL Scenario

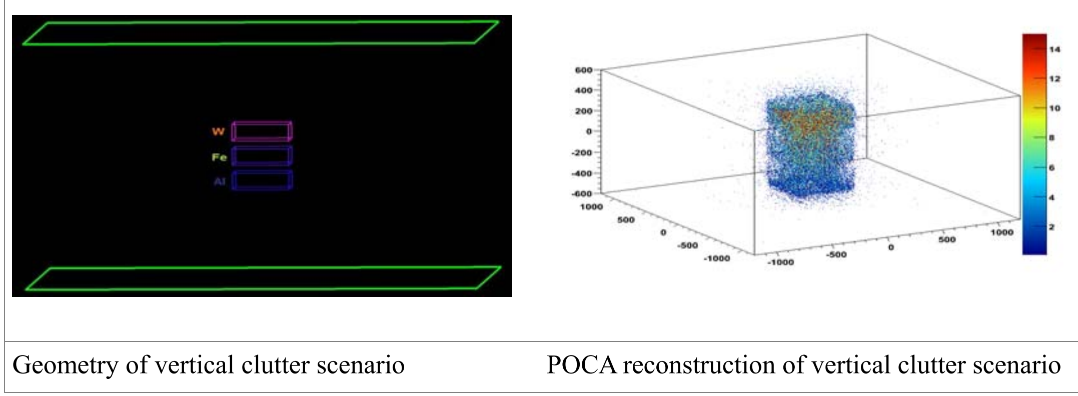


Geometry of LANL scenario	POCA reconstruction of LANL five target scenario

Figure 6.8: POCA reconstruction of LANL scenario

The POCA reconstruction of the LANL scenario (figure 6.8) again displays the ability of this algorithm to reconstruct simple scenarios with little material very well, as all three materials are reconstructed with the higher Z of the material getting scatter points with higher scattering angle.

6.3.4 Vertical Clutter Scenario

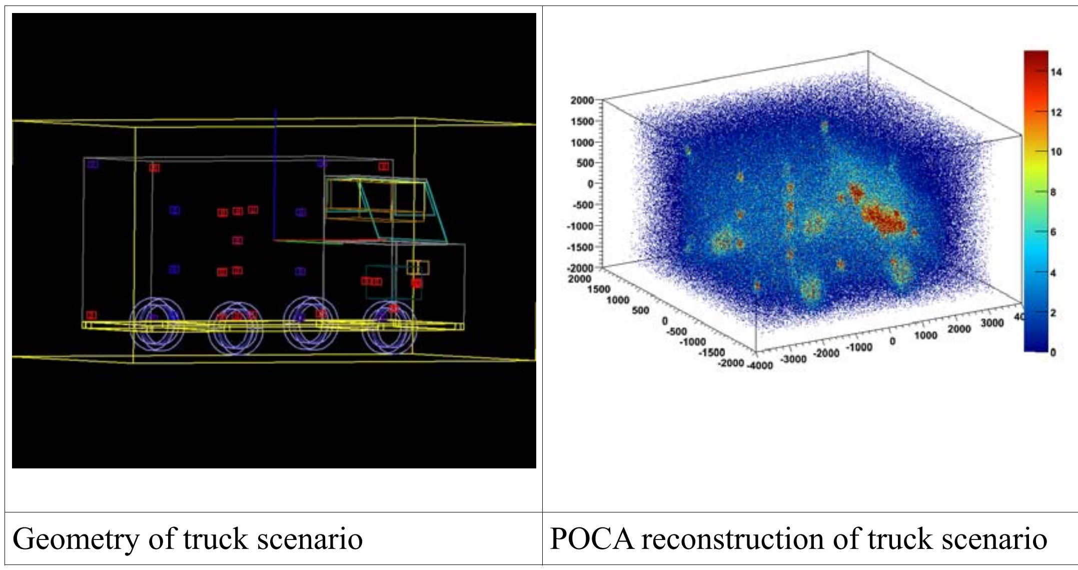


Geometry of vertical clutter scenario	POCA reconstruction of vertical clutter scenario

Figure 6.9: POCA reconstruction of vertical clutter scenario

The POCA reconstruction of the vertical clutter scenario in figure 6.9 shows one of the main problems with the algorithm. When large amount of materials are stacked vertically, the POCA algorithm tends to place many scatter points between the objects. Here the blocks of different material cannot be as easily discriminated as in the other scenarios and many large scattering muons are placed in areas where there is no material or in the iron block where high scattering should not occur as frequently.

6.3.5 Truck Scenario



Geometry of truck scenario	POCA reconstruction of truck scenario

Figure 6.10: POCA reconstruction of truck scenario

Figure 6.10 shows the POCA reconstruction of the truck scenario. This reconstruction shows both what POCA does well and does not do well. The small blocks of material that are placed around the truck without much surrounding material are reconstructed with definitive box shapes with materials having larger scattering angles according to their atomic number. However, for the targets that are surrounded (especially vertically) with much material, like the uranium blocks near the lead battery and iron engine, the reconstruction cannot discriminate them well

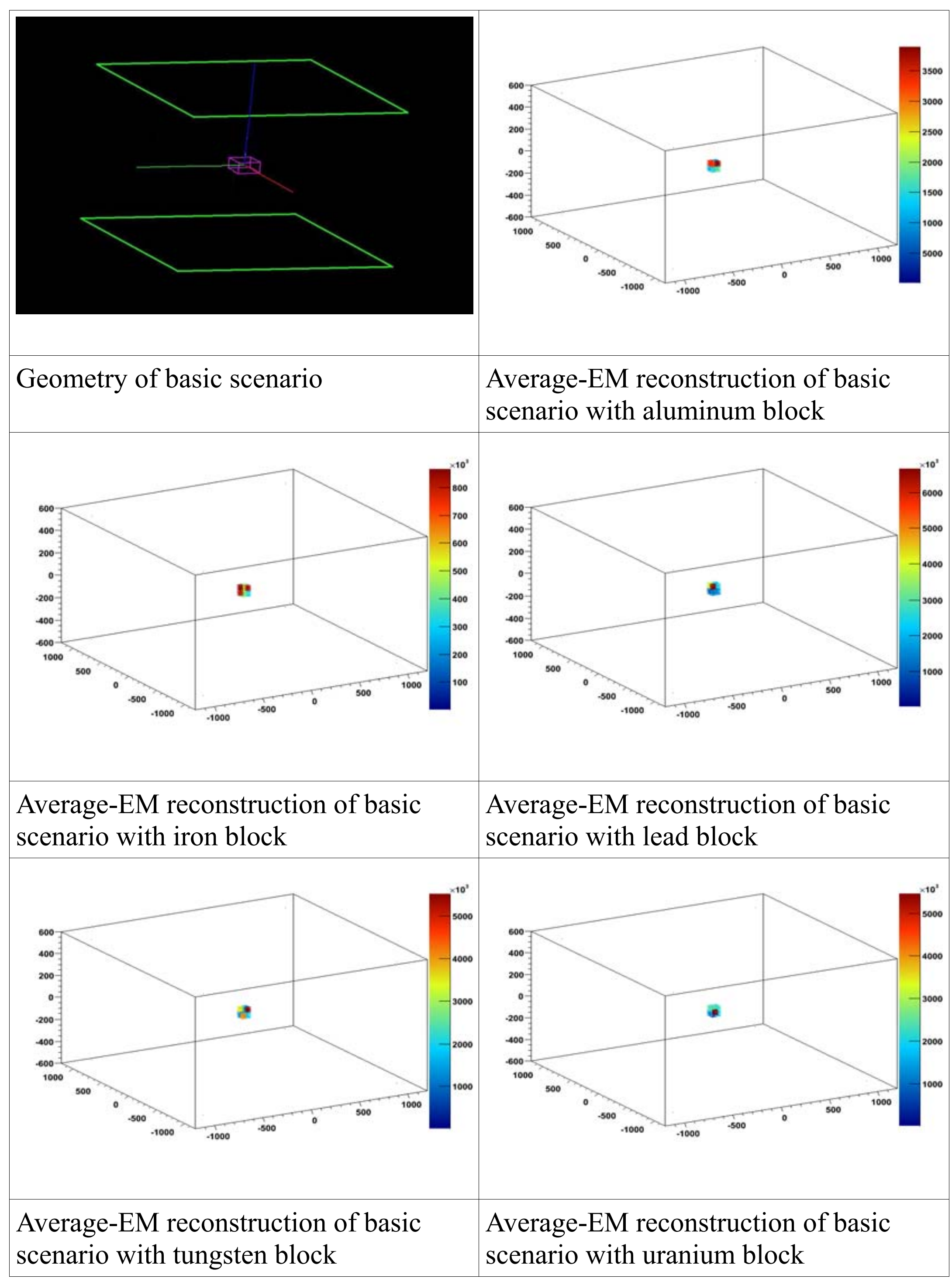
from their surroundings.

6.4 Average-EM Results

This section describes the results of the average EM method reconstruction algorithm which was described in section 4.3.2. The color scale for the plots (seen on the right side of the figures) represents the magnitude of the scattering density of the voxels in milliradians^2 per cm. The plots for the EM methods look quite different from the POCA plots as their output consists of lambda values for rectangular voxels as opposed to single points. These voxels are of a predefined size (in the following scenarios 5cm X 5cm X 5cm) and the entire voxel is colored according to its lambda value.

6.4.1 Basic Scenario

Figure 6.11 displays the results produced by the average-EM method on the basic scenarios. The reconstructions clearly show the square shapes of the objects. The absolute lambda values vary wildly, but increase in value as expected as the materials increase in atomic number.



Geometry of basic scenario

Average-EM reconstruction of basic scenario with aluminum block

Average-EM reconstruction of basic scenario with iron block

Average-EM reconstruction of basic scenario with lead block

Average-EM reconstruction of basic scenario with tungsten block

Average-EM reconstruction of basic scenario with uranium block

Figure 6.11: Average-EM reconstruction of basic scenario with different materials

Table 6.1 goes further into showing how well the targets are discriminated from their surroundings as it provides thresholds for the materials that will be used in the following scenarios to provide a base value for discrimination between objects.

Materials	λ-Threshold (mrad^2/cm)	True Positives	True Positive %	False Positives	False Positive %	False Negatives	False Negative %
Aluminum	8000	8	100	0	0	0	0
Iron	295000	8	100	0	0	0	0
Lead	800000	8	100	0	0	0	0
Tungsten	1300000	8	100	0	0	0	0
Uranium	1800000	6	75	0	0	2	5.68e-5

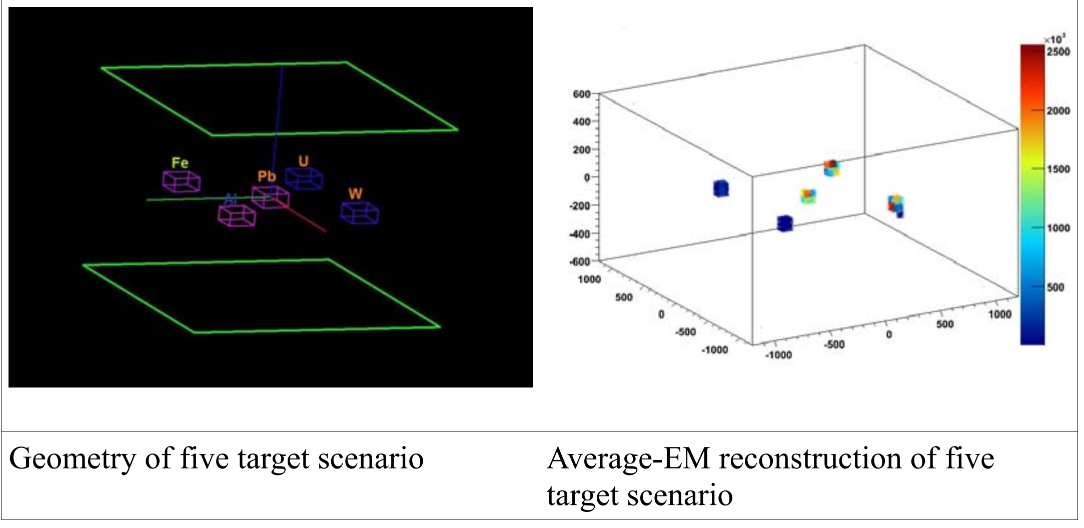
Table 6.1: Accuracy analysis of the average-EM method reconstruction of the basic scenario

Before analyzing the accuracy results, the table columns should be explained. The threshold is the chosen minimum value for a material that needs to be surpassed for a voxel to be registered as that material. True positives are the the number of voxels in the target that surpassed the threshold value. The true positive percentage is determined by dividing the number of voxels in the target that attained the threshold by the total number of voxels in the target. False positives are voxels that surpassed the threshold value that weren't in the target with the false positive percentage determined by dividing the number of voxels in the volume that surpassed the threshold by the total number of voxels in the volume. False negatives are the number of voxels in the target that did not surpass the threshold value.

The basic scenarios show excellent results in terms of accuracy for the average-EM method. Aluminum, iron, lead and tungsten all show perfect detection for the

thresholds chosen without incurring any false positives. Uranium had one voxel reconstructed with a relatively low value of lambda and one with a value near the tungsten threshold, preventing it from getting perfect detection although with its chosen threshold it also avoided any false positives. A large difference in threshold values for the materials even for tungsten and uranium, which is one of the tougher pair of materials to distinguish between, as they are both high-Z and are very close in density. However, now we have a baseline to distinguish between the materials. Also, as was mentioned earlier, the simulations are run in vacuum. Any voxel not containing material will instead have in it the density equivalent of the galactic background. These voxels will be considered reconstructed correctly if they have a value less than one, as very little scattering should occur through these voxels.

6.4.2 Five Target Scenario



Geometry of five target scenario	Average-EM reconstruction of five target scenario

Figure 6.12: Average-EM reconstruction of five target scenario

The 3D image of the five target scenario is recreated well as seen in figure 6.12. The boxlike shape of the objects are apparent and the values of lambda appear appropriate in terms of the the higher Z materials getting higher values. The accuracy of the scenario is displayed in Table 6.2.

Material	λ-Threshold ($mrad^2/cm$)	True Positives	True Positive %	False Positives	False Positive %	False Negatives	False Negative %
High-Z	800000	16	66.6	0	0	8	2.27e-4
U	1800000	5	62.5	4	0.00012	3	8.52e-5

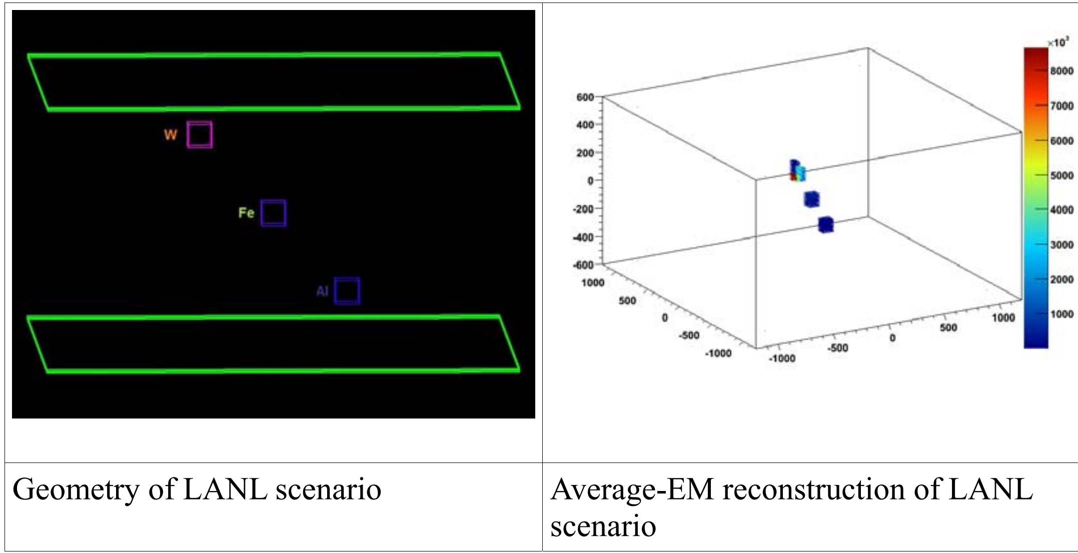
Table 6.2: Accuracy analysis of the average-EM method reconstruction of the five target scenario

The five target scenario doesn't hold up as well in terms of the voxels obtaining the thresholds chosen with the basic scenarios. Only two-thirds of the voxels were reconstructed with at least a lambda value of 800000 – the threshold for lead, the lowest high-Z material modeled. However the discrimination still exists as there were no false positives and if the threshold is lowered to 350000, all voxels containing high-Z material are present with zero false positive values. Still, this is not the ideal result as it would be best if a material had a single threshold value for lambda that would remain constant and be able to be used regardless of scenario.

The result of the accuracy for uranium is uneven as more than half of the voxels are over the threshold, but a few of the voxels from the lead and tungsten targets are over the the threshold as well. This scenario shows the difficulty of distinguishing between similar materials (lead, tungsten and uranium are high-Z and very dense), but that overall uranium produces higher scattering densities than the others.

Another noticeable feature of this scenario is the lack of noise. There are 40 total voxels out of 32000 in the reconstruction that contain material. All forty had lambda values above one and only one voxel without material got a value above one, and it was by a very slight amount. Overall this reconstruction did very well in terms of placing material in the appropriate voxels.

6.4.3 LANL Scenario



Geometry of LANL scenario	Average-EM reconstruction of LANL scenario

Figure 6.13: Average-EM reconstruction of LANL scenario

Like the previous scenarios the 3D reconstruction of the LANL simulation looks almost identical to the original geometry as seen in figure 6.13. The definition of the targets is easily seen and the image contains almost no noise. The tungsten target gets the higher lambda values with the iron and aluminum lambda values being much smaller. Table 6.3 details the accuracy of the reconstruction.

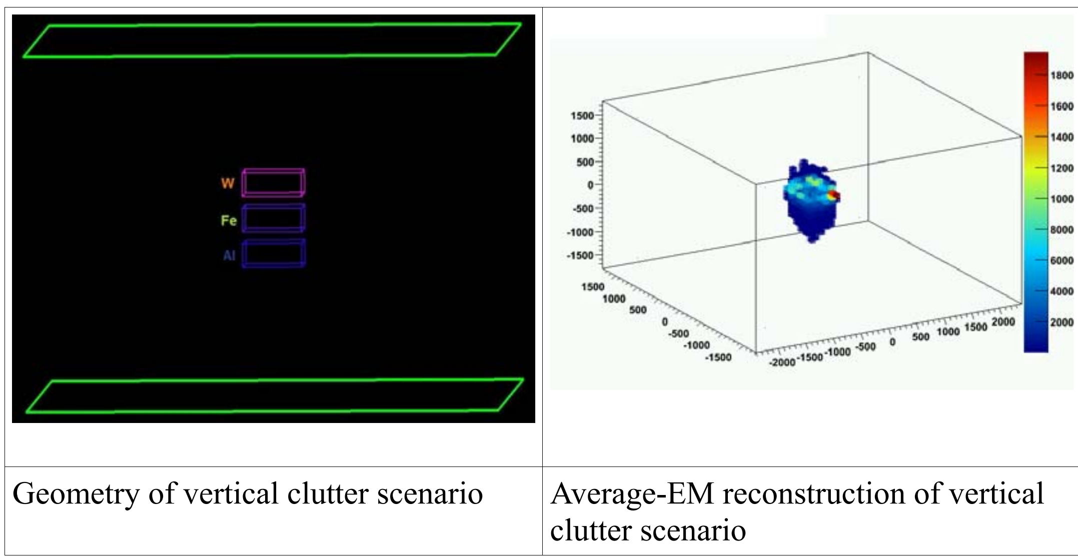
Material	λ-Threshold ($mrad^2/cm$)	True Positives	True Positive %	False Positives	False Positive %	False Negatives	False Negative %
W	1300000	6	75	0	0	2	5.68e-5

Table 6.3: Accuracy analysis of the average-EM method reconstruction of the LANL scenario

The LANL scenario holds up better in terms of accuracy than the five target scenario. Here three fourths of the tungsten voxels are reconstructed above the

threshold determined for the element with zero of the voxels being mistakenly classified as being tungsten. Only two voxels – both right above the tungsten object – that contain no material receive a lambda value above one, so this scenario is reconstructed with very little noise.

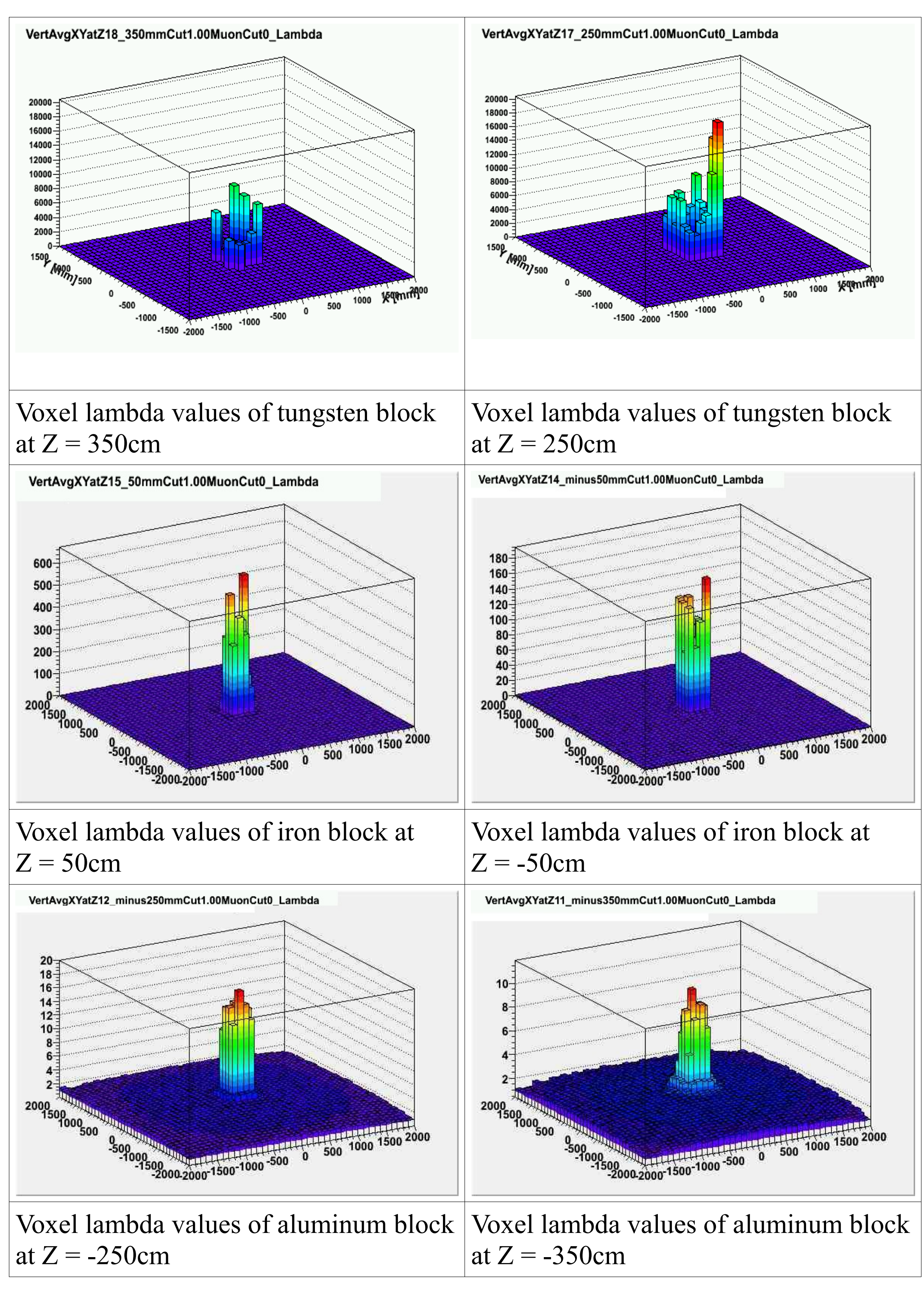
6.4.4 Vertical Clutter Scenario



Geometry of vertical clutter scenario	Average-EM reconstruction of vertical clutter scenario

Figure 6.14: Average-EM reconstruction of vertical clutter scenario

In the 3D plot found at the top of figure 6.14, it is evident that the tungsten block is reconstructed in the proper place, though noise can be seen around the block. Both the iron and aluminum targets are harder to make out, but the lego plots in figure 6.15 show that they are being reconstructed properly as well.



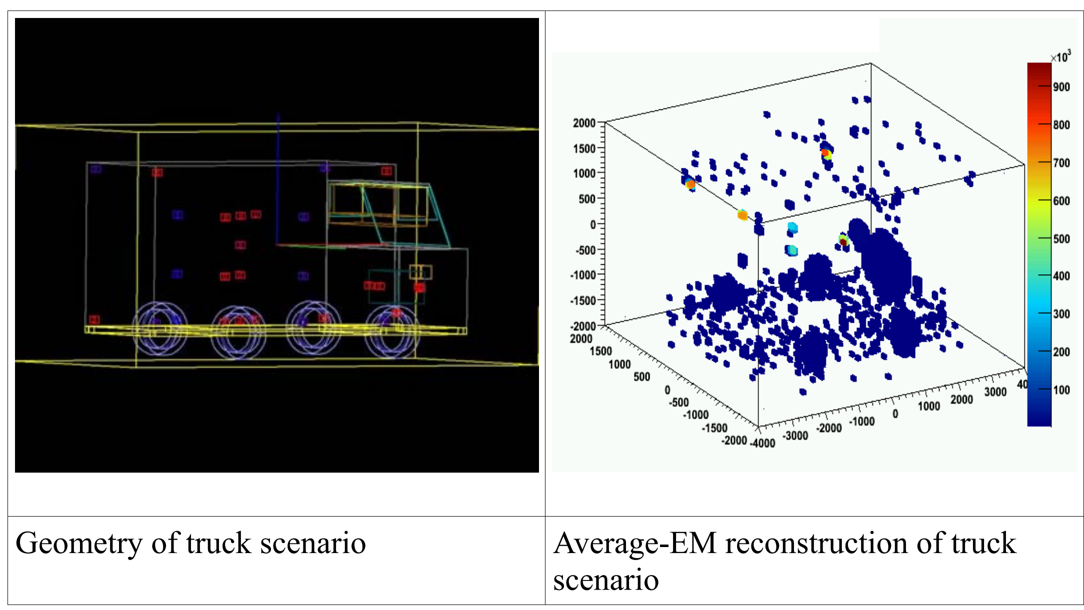
Voxel lambda values of tungsten block at Z = 350cm	Voxel lambda values of tungsten block at Z = 250cm
Voxel lambda values of iron block at Z = 50cm	Voxel lambda values of iron block at Z = -50cm
Voxel lambda values of aluminum block at Z = -250cm	Voxel lambda values of aluminum block at Z = -350cm

Figure 6.15: Lego plots of the average-EM reconstruction of the vertical clutter scenario

The plots in figure 6.15 are slices of the volume in the XY plane at different Z. Since the voxel size is 10cm X 10cm X 10cm and the blocks are 20cm thick, the blocks contain two voxels in the Z direction. The plots on the left show the value of the voxels on the higher level of the block, and the plots on the left show the value of the voxels on the lower level. Judging by these it is apparent that the algorithm 'finds' the different materials. The tungsten voxels all have higher values than the iron voxels, and the iron voxels all have higher lambdas than the aluminum voxels. However, 150 voxels have material present in them, yet 354 get reconstructed with a lambda value above 1, which was established before as a cutoff between the background vacuum and some type of material. The average-EM method seems to be able to discriminate well in this situation, but does also produce a lot of leakage into voxels adjacent to the block.

This scenario used larger voxels than the other due to the larger sized objects. In experiments it has been observed that the size of the voxel chosen can alter the value of lambda. Due to this the lego plots were used in lieu of of the accuracy analysis of the scenarios.

6.4.5 Truck Scenario



Geometry of truck scenario	Average-EM reconstruction of truck scenario

Figure 6.16: Average-EM reconstruction of truck scenario

The reconstruction of the truck scenario is one of the more obvious differences that can be seen between the average-EM method and the median-EM method. Figure 6.16 shows that the reconstruction constructed the truck well. The tires, engine, and battery are visible. Also, the targets themselves were generally found at the right spots. What's not quite as distinct is the discrimination between the different materials. Also, the reconstruction is very noisy. Table 6.4 makes these issues more apparent.

Material	λ-Threshold ($mrad^2/cm$)	**True Positives**	True Positive %	False Positives	False Positive %	False Negatives	False Negative %
High-Z	800000	**52**	36.1	0	0	92	1.23e-4
U	1300000	**33**	34.4	8	8.80e-7	69	9.20e-5

Table 6.4: Accuracy analysis of the average-EM method reconstruction of the truck scenario

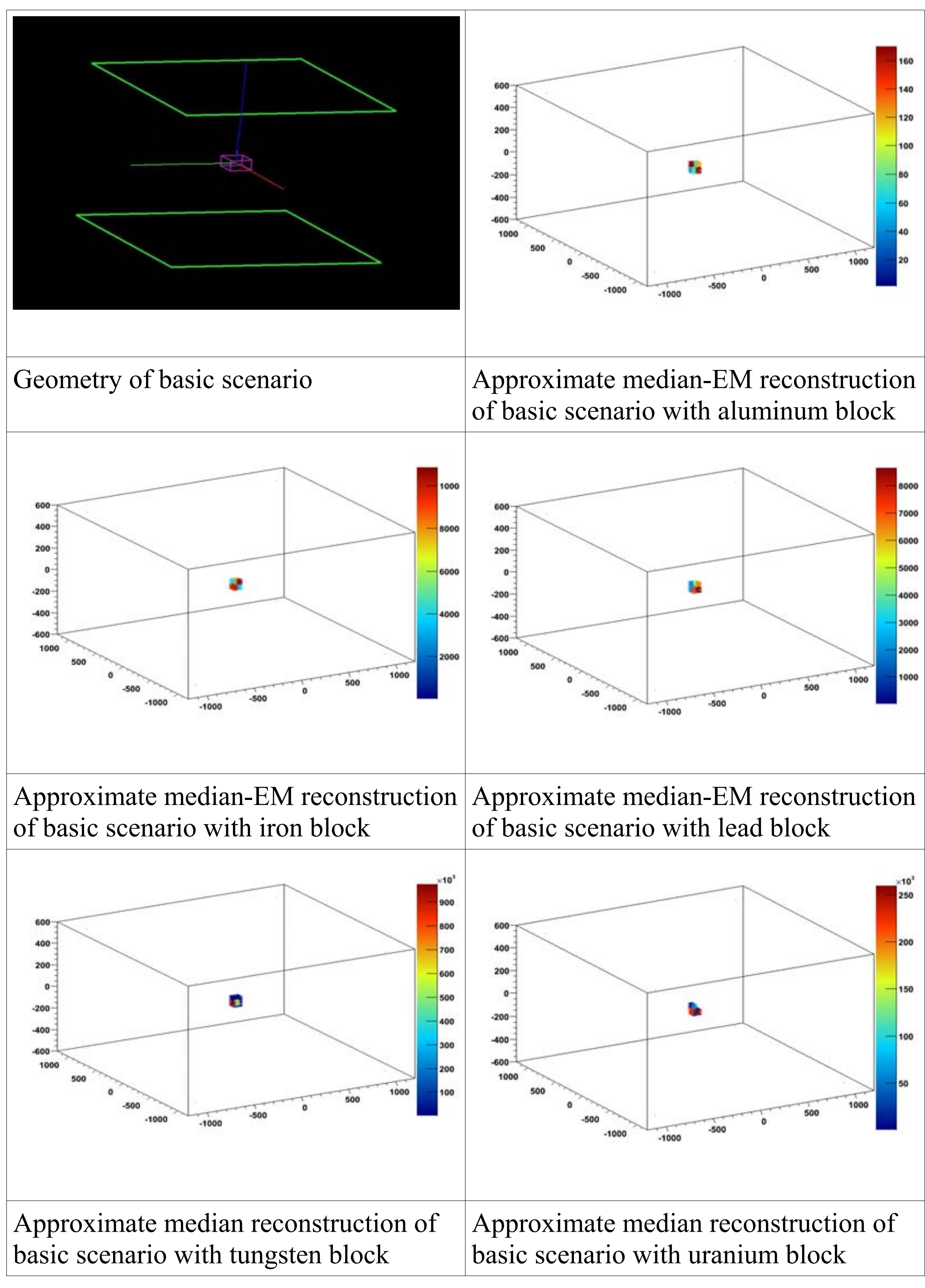
With the thresholds chosen, less than half of the voxels are reconstructed correctly for the high-Z and uranium analysis. As seen in the five target scenario, the high-Z voxels have higher values than all the voxels of lower Z material, but their absolute values are lower than the threshold chosen. However, no voxels that do not contain high-Z material are reconstructed as if they did, keeping the false positive rate at zero. For the uranium threshold, some tungsten and lead voxels do get larger than expected values and thus show up as uranium. Overall there still is discrimination between materials, but the lambda values for these materials are not staying as constant between scenarios as we would like them to.

6.5 EM Approximate Median Results

This section displays the results of the approximate EM method reconstruction algorithm. The color scale for the plots (seen on the right side of the figures) represents the magnitude of the scattering density of the voxels in milliradians^2 per cm.

6.5.1 Basic Scenario

Figure 6.17 displays the results produced by the approximate median method on the basic scenarios. Like the average method, the reconstructions clearly show the square shapes of the objects.



Geometry of basic scenario	Approximate median-EM reconstruction of basic scenario with aluminum block
Approximate median-EM reconstruction of basic scenario with iron block	Approximate median-EM reconstruction of basic scenario with lead block
Approximate median reconstruction of basic scenario with tungsten block	Approximate median reconstruction of basic scenario with uranium block

Table 6.17: Approximate median-EM reconstruction of basic scenario with different materials

Unlike the average method, the absolute lambda values vary less, as seen in table 6.5.

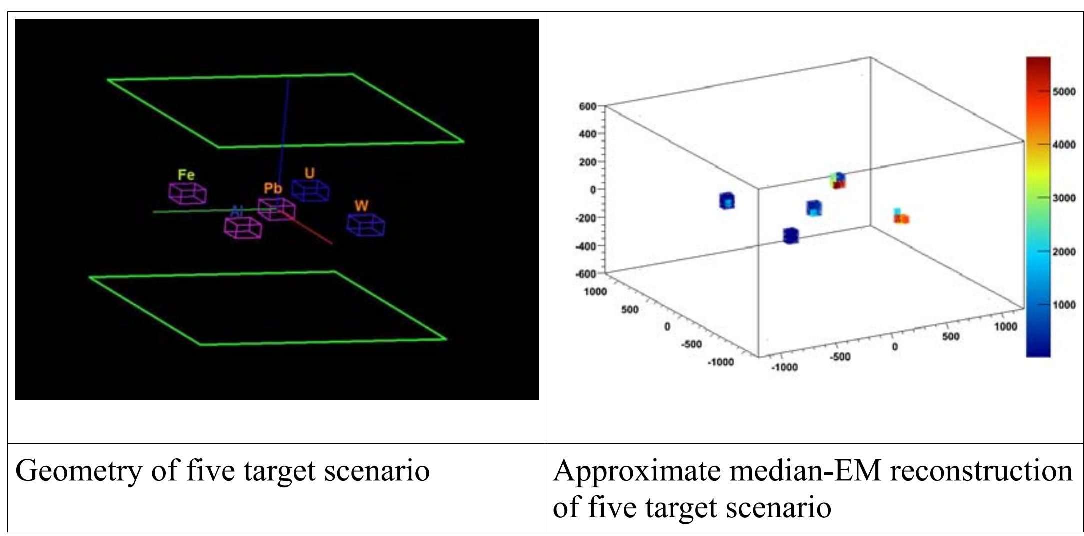
Material	λ-Threshold (mrad^2/cm)	True Positives	True Positive %	False Positives	False Positive %	False Negatives	False Negative %
Al	38	8	100	0	0	0	0
Fe	1500	8	100	0	0	0	0
Pb	2000	8	100	0	0	0	0
W	5000	7	87.5	0	0	1	3.08e-5
U	20000	7	87.5	0	0	1	3.08e-5

Table 6.5: Accuracy analysis of the approximate median-EM method reconstruction of the basic scenario

The basic scenarios show similar results in terms of accuracy for the approximate median method. No false positive voxels was incurred for any of the thresholds found. Aluminum, iron and lead were discriminated perfectly, while tungsten and uranium both had one voxel receive a low value of lambda with respect to the chosen threshold. A large difference in threshold values for the materials is apparent in this method as well and the difference for tungsten and uranium is even more pronounced. Now with the baselines established for this method, the other scenarios can be analyzed and a good comparison of the accuracy between the methods can be made.

6.5.2 Five Target Scenario

The 3D image of the five target scenario is again constructed well as seen in figure 6.18. Like the average method, the block shape of the objects can be seen and the lambda values appear correct in comparison of the different materials. The accuracy of the scenario is displayed in table 6.8.



Geometry of five target scenario	Approximate median-EM reconstruction of five target scenario

Figure 6.18: Approximate median-EM reconstruction of five target scenario

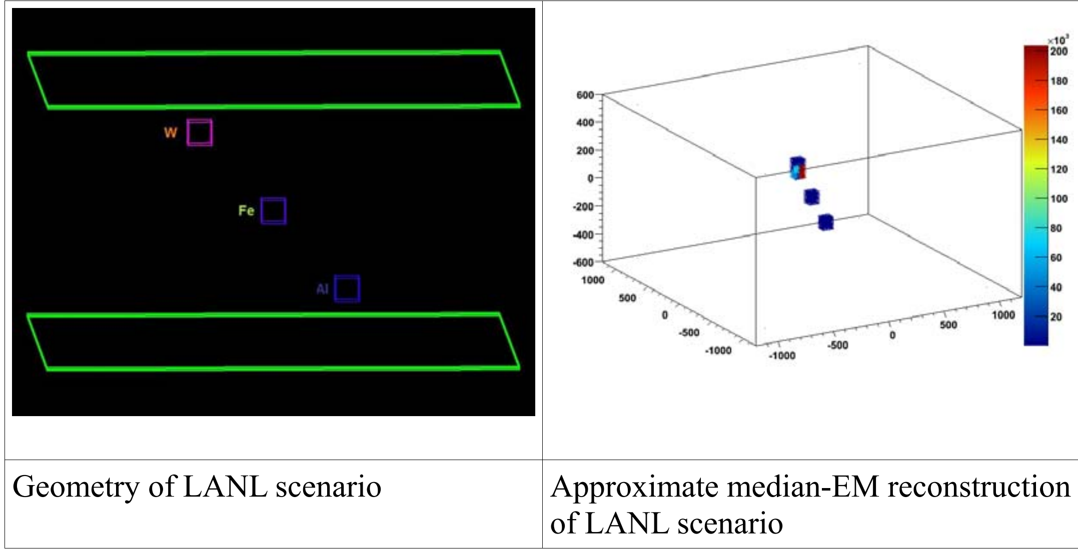
Material	λ-Threshold ($mrad^2/2$)	True Positives	True Positive %	False Positives	False Positive %	False Negatives	False Negative %
High-Z	2000	20	83.3	0	0	4	1.23e-4
U	20000	7	82.5	2	0.00006	0	0

Table 6.6: Accuracy analysis of the approximate median-EM method reconstruction of the five target scenario

Here the median method improves upon the average. For discrimination of high Z material the median method reconstructs four more voxels correctly than the average method for an 83.3% detection rate with zero false positives. All high Z voxels have lambda values above all non high Z voxels as well, so total discrimination remains perfect. The detection of uranium also improves for the median method as all but one voxel exceeds the threshold, while the average method had three misses. The false positives also decreased by half from four for the average method to two for the median.

For the first comparison scenario it appears that the median method improved upon the average. The effect of the non-Gaussian scattering is lessened by using the approximate median-EM as opposed to the average-EM, and the high variance of lambda values from voxel to voxel is not as readily seen with this method.

6.5.3 LANL Scenario



Geometry of LANL scenario	Approximate median-EM reconstruction of LANL scenario

Figure 6.19: Approximate median-EM reconstruction of LANL scenario

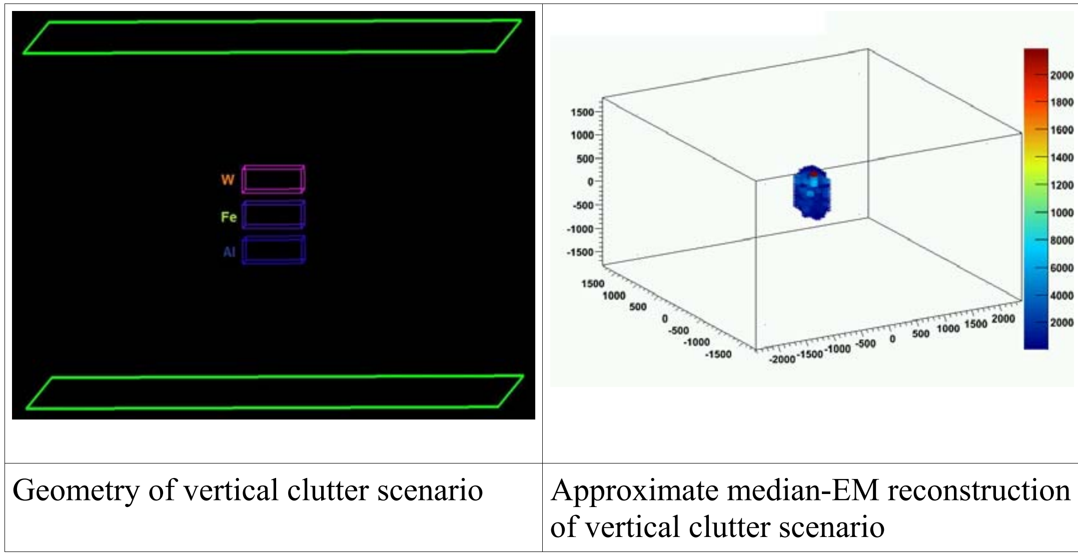
Figure 6.19 shows that the 3D image produced looks almost identical to the geometry and contains minimal noise. The tungsten target gets higher lambda values than iron and aluminum, and the iron is higher than aluminum. Table 6.7 shows the accuracy of the reconstruction.

Material	λ-Threshold ($mrad^2/cm$)	True Positives	True Positive %	False Positives	False Positive %	False Negatives	False Negative %
W	5000	8	100	0	0	0	0

Table 6.7: Accuracy analysis of the approximate median-EM method reconstruction of the LANL scenario

The discrimination of tungsten versus the other materials with the threshold found is perfect. All voxels passed the threshold and no voxels containing material other than tungsten did. There were two voxels above the tungsten block that did get lambda values above one, but they were very low values. Little noise overall appeared in this scenario, and like the five target scenario, the discrimination for the median method appears superior to the average method which only constructed 75% of the tungsten voxels above the threshold found.

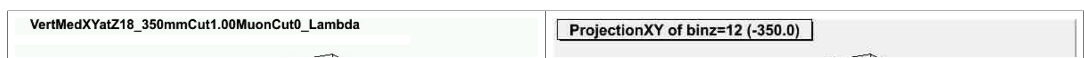
6.5.4 Vertical Clutter Scenario

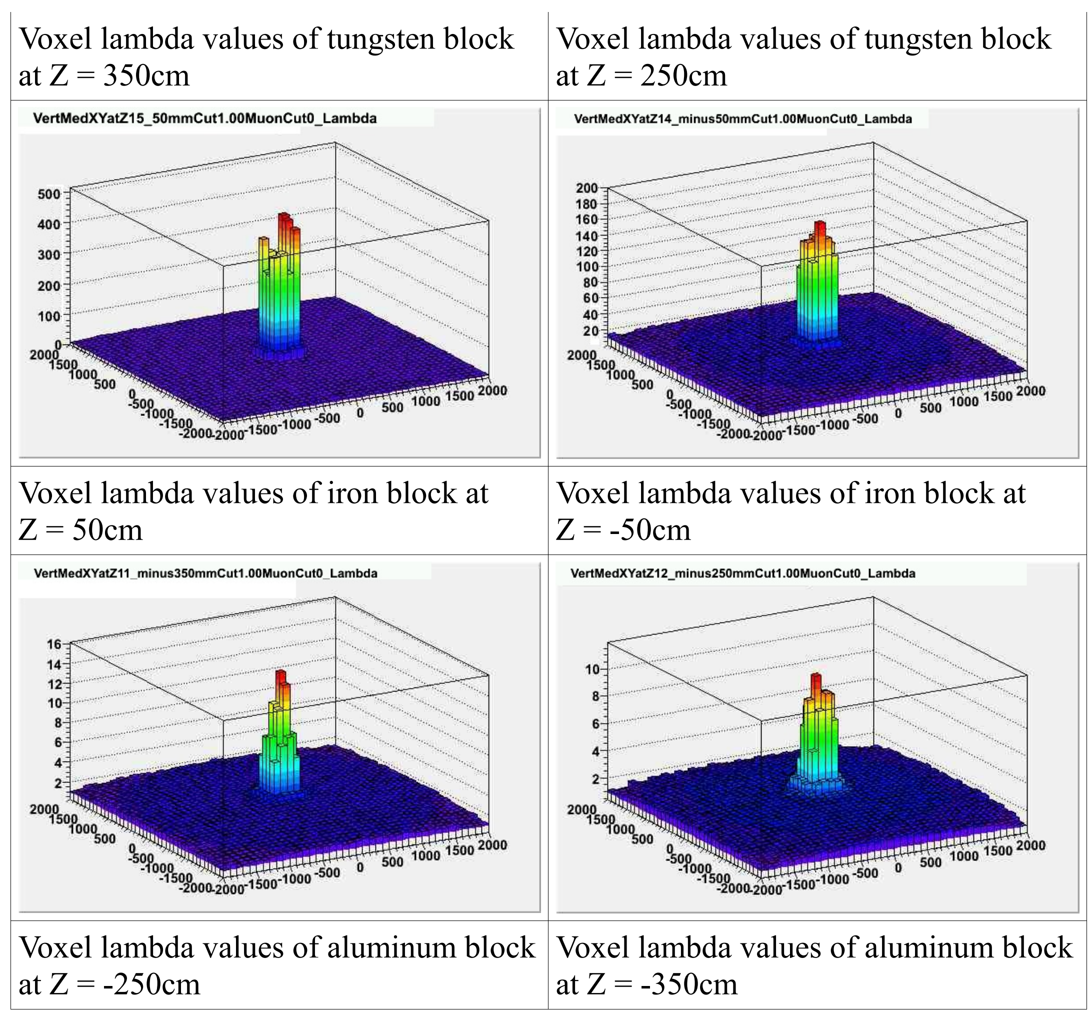


Geometry of vertical clutter scenario	Approximate median-EM reconstruction of vertical clutter scenario

Figure 6.20: Approximate median-EM reconstruction of vertical clutter scenario

Much like the average method, judging by the 3D plot found at the top of Figure 6.20, it is evident that the tungsten block is reconstructed correctly, though discrimination between the boxes is not easily seen. However, unlike for average method much less noise is seen in the plot as well. The lego plots in figure 6.21 show definite discrimination between the materials.





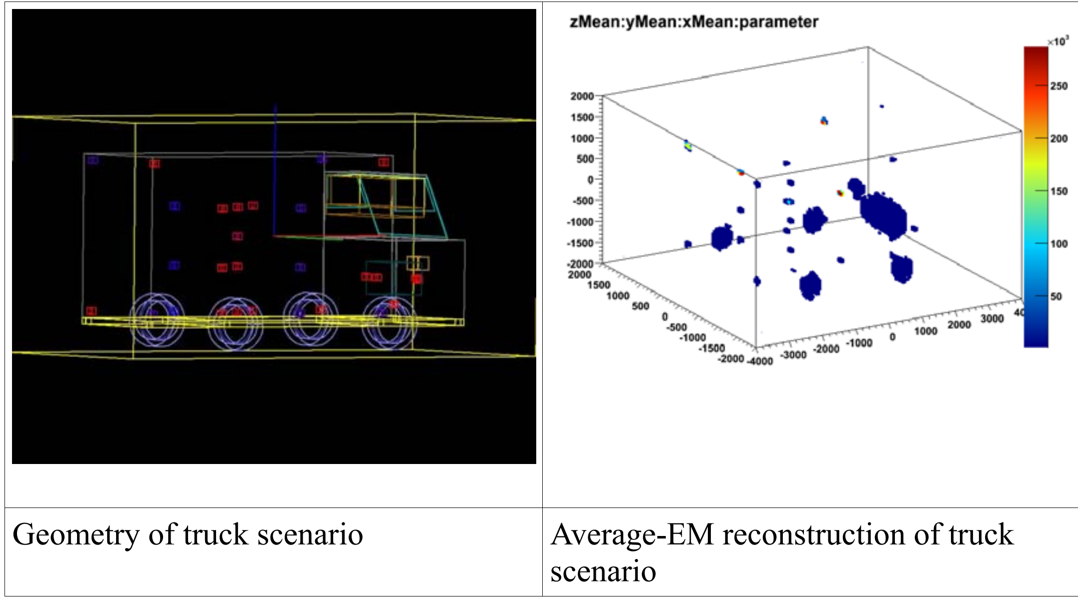
Voxel lambda values of tungsten block at Z = 350cm	Voxel lambda values of tungsten block at Z = 250cm
Voxel lambda values of iron block at Z = 50cm	Voxel lambda values of iron block at Z = -50cm
Voxel lambda values of aluminum block at Z = -250cm	Voxel lambda values of aluminum block at Z = -350cm

Figure 6.21: Lego plots of the approximate median-EM reconstruction of the vertical clutter scenario

The lego plots in figure 6.21 show the tungsten voxels having higher lambda values than the iron voxels, and the iron voxels higher lambda values than the aluminum. In comparison to POCA both the average and median methods of EM do well in discriminating between materials in scenarios with vertical clutter. Between the two EM algorithms though, the median method does a much better job of leakage control. The average method had 204 voxels reconstructed with a lambda value above 1 that contained no material. The approximate median method only reconstructed 17 of such voxels. As the average method, the approximate median

seems to be able to discriminate well even with vertical clutter, but does so with producing much less leakage.

6.5.5 Truck Scenario



Geometry of truck scenario	Average-EM reconstruction of truck scenario

Figure 6.22: Approximate median-EM reconstruction of truck scenario

Figure 6.22 shows the 3D image of the truck scenario reconstruction for the approximate median. The improvement the approximate median offers over the average-EM method is very obvious with this scenario. All car components are reconstructed properly and the targets are clearly seen as well. What is also different from the average-EM method, is the total lack of leakage. The targets are distinct and can be clearly distinguished from the background; something that was difficult to do in the 3D image of the average method reconstruction. Table 6.8 shows the improvement of the accuracy as well.

Material	λ-Threshold (mrad^2/cm)	True Positives	True Positive %	False Positives	False Positive %	False Negatives	False Negative %

High-Z	2000	127	88.2	0	0	17	2.27e-5
U	20000	55	57.2	0	0	41	5.46e-5

Table 6.8: Accuracy analysis of the approximate median-EM method reconstruction of the truck scenario

The accuracy rate for high-Z materials more than doubled from 33% to over 88% in changing from the average-EM to the approximate median-EM. The approximate median-EM does a much better job than the average-EM in discriminating these high-Z materials from the other objects in the truck and does so without reconstructing any voxels as high-Z that aren't. The accuracy for using the uranium threshold doesn't improve as dramatically as the high-Z threshold, but it is still a marked improvement from about 32% detection with the average-EM method to over 57% detection with the median method. Also gone are all the tungsten and lead voxels that passed the uranium threshold in the average-EM method. Overall the discrimination and cleanliness of the approximate median-EM method is far superior to the reconstruction of this scenario with the average method.

6.5 Analysis of Results

With these results from simulations for all three reconstruction methods, it has been shown that muon tomography is a feasible alternative for cargo inspection. The POCA algorithm shows very nice results for simple scenarios and gives a nice base for comparison purposes with the EM methods. However, it is not robust enough to deal with the issue of multiple scattering without additional work being done on it.

As far as the EM methods go, both show definite discrimination of high Z targets from non-high Z targets, although discrimination of uranium from other high Z materials was not quite as apparent. However, the approximate median method does a much better job overall than the average method in these regards. The discrimination of high Z from non-high Z materials and of uranium from other high Z materials is better in every scenario, and it also is vastly superior in terms of creating less noise than the average method.

It is important to note however, that despite the high level of discrimination obtained from the results based on the threshold values of lambda, the absolute values of lambda are much different than would be expected. Using the equation for scattering density shown in chapter 4, the absolute value of λ can be determined for a material based on it's radiation length and a nominal momentum. For a nominal momentum of 3 GeV, Aluminum should have a scattering density of 2.8 $mrad^2/cm$, while Iron and Tungsten should have scattering densities of 14.2 $mrad^2/cm$ and 71.5 $mrad^2/cm$ respectively. As seen in this chapter, the λ values determined by our reconstructions were much higher than the true values of λ. Experiments at LANL [7] produced λ's much closer to the actual values, though the average EM produced estimates that were too high, while the median EM gave results closer to the true values. Work is ongoing to produce scattering densities from our reconstructions that are closer to the real values. Despite this problem the differences in scattering densities between materials that were created in our experiments were enough to get clear discrimination between targets.

However, there are other issues in addition to the actual reconstruction results produced that are highly relevant to the algorithms, such as computation and memory usage. Memory consumption for the true median and approximate median methods was discussed in chapters 4 and 5, but the run times for the algorithms will

be explored in the next section.

6.6 Algorithms

The EM algorithm is computationally expensive as shown in chapter 4. The running time is influenced by the number of muon events, the number of the voxels (itself dependent on volume and voxel size), and the number of iterations the algorithm is run over. Changes to the implementation of the algorithm were made in an effort to make it more efficient as detailed in chapter 5. The development of the approximate median-EM method actually came about due to the computation and memory problems the true-median EM method brought forth. The true-median EM method requires storage of all correction values for every voxel so that they can be sorted and the median found. This mandates both a large memory and computational time increase compared to the average method as there may be millions of voxels containing thousands of muon events, all of which need to be stored and sorted.

Table 6.9 compares the running times of the implementation of both the improved average-EM and approximate median-EM algorithms, against the naive implementations of the average-EM and true median-EM algorithms. The basic scenario described in section 6.2.1 is used for the tests and is run with uranium. This scenario is of a 10cm X 10cm X 10cm box placed in the center of a 200cm X 200cm X 110cm volume. The voxel size is 5cm X 5cm X 5cm. 10 minutes exposure time was used (1,000,000 muon events) and 100 iterations were run. Twenty runs were made (except for the true median-EM method) for each algorithms and the times averaged to ensure accurate results and to account for anomalous high or low run times:

Algorithm	**Time (Average of 20 runs except for**

	true median)
Average-EM (LANL implementation)	523 seconds
Average-EM (improved implementation)	316 seconds
Approximate Median-EM	1173 seconds
True-median EM	25.5 hours

Table 6.9: Timing comparison of the different implementations and methods of the EM algorithm

Both methods used in this study run much quicker than the naive methods. The average-EM of this study saves almost 40% computation time against its counterpart. The results of the approximate median are even more stark. When the true median method was originally implemented using insertion sort, it took twenty-five and a half hours to complete. This necessitated the development of the approximate median method which runs exceptionally faster than the true median implementation. The changes made in the implementation of the EM algorithm appear to have significantly improved the running times of both methods. However, despite its better reconstruction results, the approximate median-EM lags far behind the average-EM in terms of runtime.

Chapter 7
Conclusions and Future Work

7.1 Summary

This thesis explored the current state of cosmic ray muon tomography, detailed the improvements and additions to existing reconstruction algorithms and their implementation, and showed results from the POCA, average-EM and new approximate median-EM methods. Chapter 1 explained the purpose of muon tomography and its importance, which included the description of its advantages over other techniques. Chapter 2 gave much background information about muon tomograpy: it defined what tomography is and how tomographic reconstruction generally works, showed what muons are and the physics pertaining to their passage through matter, discussed how the physics are used to form the concept of muon tomography, and detailed the past work done with muons and other emission tomography for imaging purposes. Chapter 3 described pertinent research questions and laid out the expectations of the study as well as the additions it could potentially make to the field. Chapter 4 took a thorough look at the reconstruction algorithms used in this study by explaining how the POCA, average-EM, true median-EM, and approximate median-EM methods work, as well as analyzing the advantages and disadvantages each method possesses. In Chapter 5 was an overview of the tools used for the entire process from simulation to reconstruction. It also gave a detailed look into the implementation of the muon tomography suite and the software testing techniques used to debug it. Lastly, Chapter 6 displayed the results obtained from the reconstruction algorithms run on different scenarios as well as an analysis of the run times of the EM methods.

It was concluded that the use of muon tomography for cargo inspection is definitely

a feasible idea and that our implementation of the approximate median-EM algorithm does a better job of discriminating between materials than the simplistic POCA and average-EM methods, and does so without the poor running time of the true median-EM algorithm.

7.2 Future Work

In the course of research many ideas for improvements or additions to the POCA and EM algorithms were suggested. Some were even implemented, but because of time restraints weren't tested extensively. This final section details these developments and the future work that can be done to continue the research found in this thesis.

7.2.1 Real Time EM

The current status of the EM algorithm for muon tomography is that it needs all input data before the iterations between the expectation and maximization steps begin. Also, there is no output until the max iterations are reached or the lambda values converge. This means that no analysis of the reconstruction can be done until the algorithm finishes. POCA on the other hand can be updated after every event is processed. Finding a way to add this feature to EM would be helpful as it would speed up reconstruction and on-the-fly analysis of a volume could be done.

The idea proposed here is to run the EM algorithm in parallel. One thread would continually collect and process the input data while another thread would run the expectation and maximization steps on the already collected idea. Experimentally a threshold would be determined for how much data to collect before passing it to the EM algorithm, and whenever this threshold is reached a signal would be sent to the EM telling it to stop after its next iteration, output the current lambda values and collect the next batch of data. A similar concept was explored by Ahn, et al. [35]

that used ordered subsets to speed up maximum likelihood estimations. These types of algorithms have been developed in the past, but had problems with convergence until the approach by Ahn had been developed [35].

The MTS is not implemented on parallel architecture so a true parallel EM that runs in real time could not be tested. However, a way to simulate the process was created instead. While the data collection and EM are not run simultaneously, a predetermined set of events are processed at which point the EM begins to run. After a set number of iterations the EM stops, but the lambda values of the voxels and the already received data is saved. The current lambdas are output and then more data is collected and passed back to the EM. The algorithm continues until all the data are received. Very few tests were run on this real time EM, and the lambda values did not converge well as has been the case with similar approaches [35]. However, the testing was not thorough and the real time EM seems to be a good avenue of research for potential improvements to the reconstruction algorithms.

7.3.2 POCA/EM Integration

POCA provides good results in simple scenarios and is already used in our implementation as a base for EM to predict a muon track through a volume. Other ways were considered to use POCA to improve the performance of the EM algorithm.

One such way was to bias voxels based on their distance from the scatter point. Several ways exist to do this. Instead of considering all voxels a muon track passed, through, use only the voxel that the scatter point was reconstructed in. Alternatively, the voxels can be weighted based on how far they are from the voxel containing the scatter point. This technique was implemented but not extensively tested. It is another area that could provide a performance increase to EM if

researched is continued.

Several other techniques based on POCA have been proposed and were possibilities for study by the author but not implemented, and are actually being worked on by other students at Florida Tech. One idea is filtering events based on the distance of closest approach (DOCA) between the incoming and outgoing tracks. Another attempt is being made at clustering the scatter points obtained from POCA and using them to either calculate lambda values directly or running EM only on the voxels contained in the clusters. Initial results have been promising and the decrease in running time has been drastic.

7.3.2 Other Work

Besides improvements to reconstruction algorithms there is much work to be done continuing the research in this study. So far all of the reconstruction work has been done on simulations from Geant4. The High Energy Physics lab at Florida Tech is currently developing GEM detectors to be used in muon tomography systems. Eventually the reconstruction algorithms from this study will be used on real world data produced from these systems. Also, until the detectors are finished more detailed analysis can be done using simulations, such as testing the effect of changing parameters in the reconstruction algorithms (like initial lambda) or implementing input filters (like momentum or angle cuts). In addition, more realistic simulations can be tested in which the scenarios don't have ideal conditions and detector resolution becomes an issue.

REFERENCES

4.4 M. Lewis, E. Shapiro, "A Novel Method of Detecting Shielded Nuclear Weapons and Voids in Cargo," *MU-VISION, INC., PROPRIETARY*, February 2008, pp. 1-15.

4.5 Y. Vardi, et al., "A Staistical Model for Positron Emission Tomography," *Journal of the American Statistical Association* 80(389), March 1985, pp. 8-20.

4.6 P. J. Green, "Bayesian Reconstructions From EmissionTomography Data Using a Modified EM Algorithm," *IEEE Trans. Medical Imaging*, vol. 9 (1), March 1990, pp. 84-93.

4.7 I. A. Elbakri, J. A. Fessler, "Statistical Image Reconstruction for Polyenergetic X-Ray Computed Tomography," *IEEE Trans. Medical Imaging*, vol. 21 (2), February 2002, pp. 89-99.

4.8 R. van Grieken, K. Janssens, *Microfocus x-ray computed tomography (mCT) for archaeological glasses -- T. Doménech Carbó*, Cultural Heritage Conservation and Environmental Impact Assessment by Non-Destructive Testing and Micro-Analysis, (2004).

4.9 W. Munk, P. Worcester, C. Wunsch, *Ocean Acoustic Tomography,* Cambridge University Press., New York, NY (1995).

4.10 K. Hagiwara, et al., Particle Data Group, Review of Particle Physics, *Phys. Rev. D* 66(1), 2002.

4.11 Williamson Labs. WMD Hunting Technology. (2005). [Online] Available: www.williamson-labs.com/ltoc/cbr-tech.htm

4.12 O.C. Allkofer, P.K.F. Grieder, "Cosmic Rays on Earth," *Fach-informations-zentrum Energie, Physik, Mathematik*, Nr 25-1 (1984).

4.13 Cosmic-ray Physics Team Lawrence Livermore National Laboratory, "Monte Carlo Simulation of Proton-induced Cosmic-ray Cascades in the Atmosphere," *UCRL-TM-229452*, February 2008.

4.14 L. Schultz, "Cosmic ray muon radiography," Ph.D. dissertation, Portland State University, 2003.

4.15 H. Bethe, "Moliere's theory of multiple scattering," *Physical Review*, vol. 89(6), 1953, pp. 1256.

4.16 L. Cox, et al., "Detector Requirements for a Cosmic Ray Muon Scattering Tomography System," *in 2008 IEEE Nuclear Science Symposium Conference Record,* November 2008, pp. 1278-1284.

4.17 J. Green et al., "Optimizing the tracking efficiency for cosmic ray muon tomography," *in 2006 IEEE Nuclear Science Symposium Conference Record*, October 2006, pp. 286–288.

4.18 F. Sauli, "GEM: A new concept for electron amplification in gas detectors," *Nucl. Instrum. Meth. A,* vol. 386, 1997, pp. 531-534.

4.19 CERN Courier. A GEM of a Detector, (1998). [Online]. Available: http://cerncourier.com/cws/article/cern/27921.

4.20 K. Gnanvo, et al., "Performance Expectations for a Tomography SystemUsing Cosmic Ray Muons and Micro Pattern Gas Detectors for the Detection of Nuclear Contraband," i*n 2008 IEEE Nuclear Science Symposium Conference Record*, November 2008, pp. 1278-1284.

4.21 M. Hohlmann, "DoD Review," HEP @ Florida Tech, 2008

4.22 L. J. Schultz, et al., "Statistical reconstruction for cosmic ray muon tomography," *IEEE Trans. Image Processing*, vol. 16 (8), 2007, pp. 1985-1993.

4.23 E.P. George, "Cosmic Rays Measure Overburden of Tunnel," *Commonwealth Engineer*, July 1, 1955, pp 455-457 (1955).

4.24 L.W. Alvarez, et al., "Search for Hidden Chambers in the Pyramids," *Science* 167, pp 832-839 (1970).

4.25 K. Nagamine, "Geo-tomographic Observation of Inner-structure of Volcano with Cosmic-ray Muons," *Journal of Georgraphy,* 104(7), pp 998-1007 (1995).

4.26 S. Minato, "Feasibility of Cosmic-Ray Radiography: A Case Study of a Temple Gate as a Testpiece," *Materials Evaluation* 46, pp 1468-1470 (1988).

4.27 E. Frlez, et al., “Cosmic Muon Tomography of Pure Cesium Iodide Calorimeter Crystals,” *Nuclear Instruments and Methods in Physics Research A* 440, pp 57-85 (2000).

4.28 K. Borozdin, G. Hogan, C. Morris, W. Priedhorsky, A. Saunders, L. Schultz, and M. Teasdale, “Radiographic imaging with cosmic-ray muons,” *Nature*, vol. 422, p. 277, March 2003.

4.29 N. Hentgartner, et al., “Enabling Port Security Using Passive Muon Radiography,” *Statistical Science Group, Los Alamos National Laboratory*, pp. 1-37.

4.30 K. Borozdin, et al., “Cosmic-ray Muon Tomography and its Application to the Detection of High-Z Materials ,” Los Alamos National Laboratory.

4.31 C. Morris, et al., “Tomographic Imaging with Cosmic Ray Muons,” Los Alamos National Laboratory, 2007.

4.32 G. Wang, J. Qi, “Statistical Image Reconstruction For Muon Tomography Using Gaussian Scale Mixture Model,” *IEEE*, 2008, pp. 2948-2951.

4.33 C. Motooka, Y. Watanabe, “Feasibility Study of Tomographic Imaging With Cosmic-ray Muons,“ Kyushu University, 2004.

4.34 Richard Hoch, Debasis Mitra, Kondo Gnanvo, and Marcus Hohlmann. "Muon Tomography Algorithms for Nuclear Threat Detection." *In Springer Lecture Series in Computational Intelligence*, No. 214, pp. 225-231, June 2009.

4.35 K. Lange, J. A. Fessler, "Globally Convergent Algorithms for Maximum a Posteriori Transmission Tomography," *IEEE Trans. Image Processing*, vol. 4 (10), 1995, pp. 1430-1438.

4.36 T. Hebert, R. Leahy, "A Generalized EM Algorithm for 3-D Bayesian Reconstruction from Poisson Data Using Gibbs Priors," *IEEE Trans. Medical Imaging*, vol. 8 (2), March 1989, pp. 194-202.

4.37 H. M. Hudson, R. S. Larkin, "Accelerated Image Reconstruction Using Ordered Subsets of Projection Data," *IEEE Trans. Medical Imaging*, March 1994, pp. 100-108.

4.38 S. Ahn, J. A. Fessler, D. Blatt, A. O. Hero, "Convergent Incremental Optimization Transfer Algorithms: Application to Tomography," *IEEE Trans. Image Processing*, vol. 25 (3), March 2006, pp. 283-296.

4.39 HEP Group. Cluster Computing, (2006). [Online]. Available: http://research.fit.edu/hep_labA/rocks/overview.html/.

4.40 D. Sunday. (2006). Distance between Lines and Segments with Their Closest Point of Approach [Online]. Available: http://geometryalgorithms.com/Archive/algorithm_0106/algorithm_0106.htm.

4.41 D.H. Eberly, *3D Game Engine Design*, Morgan Kaufmann, San Fransisco, CA (2001).

4.42 S. Teller. (2001). Computer Graphics Algorithms Frequently Asked Questions [Online]. Available: http://www.faqs.org/faqs/graphics/algorithms-faq/

4.43 W. Priedhorsky, K. Borozdin, G. Hogan, C. Morris, A. Saunders, L. Schultz, and M. Teasdale, "Detection of high-z objects using multiple scattering of cosmic ray muons," *Review of Scientific Instruments*, vol. 74, no. 10, pp. 4294–4297, October 2003.

4.44 B. Rossi, *High Every Particles*, Prentice-Hall, Inc., Englewood Cliffs, NJ (1952).

4.45 S. Eidelman et al., "Review of particle physics," *Physics Letters*, vol. B592, 2004.

4.46 J. Allison, et al., "Geant4 Developments and Applications," *IEEE Trans. Nuclear Science*, vol. 53 (1), February 2006, pp. 270-278.

4.47 S. Agostinelli et al., "Geant4: A simulation toolkit," *Nucl. Instrum. Meth. A*, vol. 506, no. 3, pp. 250–303, 2003.

4.48 RD44. Geant4 R&D Project, (1998). [Online]. Available: http://wwwasd.web.cern.ch/wwwasd/geant/rd44.html

4.49 C. Hagmann, D. Lange, D. Wright, "Cosmic-Ray Shower Generator (CRY) for Monte Carlo Transport Codes," in *2007 IEEE Nuclear Science Symposium Conference Record*, 2007, pp. 1143-1146

4.50 The ROOT Team. About ROOT, (2009). [Online]. Available: http://root.cern.ch/drupal/content/about.

Appendix

The appendix contains all files needed for the MTS suite. It also contains the makefile necessary to build the program. The files should be in a directory called “src” and the makefile should be in thedirectory containing “src”. The command 'make' can then be entered when in the the directory containing “src” and the program will be built. The programs name is “mts”. To run, first the appropriate options should be set in the file “config.tst”. To then run the program the command is './mts config.tst'.

```
//makefile for MTS
all: mts

mts: src/driver.o src/mtserr.o src/preprocessing.o src/vecfunc.o src/em.o src/mtsio.o
	g++ -lm src/driver.o src/mtserr.o src/preprocessing.o src/vecfunc.o src/em.o src/mtsio.o -o
mts

driver.o: src/driver.c src/mts.h src/preprocessing.h src/mtserr.h src/em.h
	g++ -c drive.c

mtserr.o: src/mtserr.c src/mts.h src/mtserr.h
	g++ -c src/mtserr.c

preprocessing.o: src/preprocessing.c src/mts.h src/preprocessing.h src/mtsio.h src/mtserr.h src/em.h
src/vecfunc.h
	g++ -lm -c src/preprocessing.c

vecfunc.o: src/vecfunc.c src/vecfunc.h src/mts.h
	g++ -lm -c src/vecfunc.c

em.o: src/em.c src/mts.h src/em.h src/mtsio.h
	g++ -lm -c src/em.c

mtsio.o: src/mtsio.c src/mts.h src/mtsio.h src/mtserr.h src/vecfunc.c
	g++ -c src/mtsio.c

clean:
	rm src/*.o
```

```
###		Configuration file for MTS		###
### The '#' symbol declare the proceeding line as a comment ###
### To run MTS, first enter 'make' in the same directory of makefile ###
### Then enter command - './mts config.tst' ###

### Input File - Required (output has same name but with .em extension) ###
input input/test.txt

### Voxel Sizes - Millimeters - Optional (default is 100) ###
x_voxel_size 50
y_voxel_size 50
z_voxel_size 50

#### Em - 3 Options (average, median, 3D) - All may be run concurrently ###
em average
em median
#em 3D

#### Bin Info for Median EM (default number of bins are 100 and default size per bin is 10000) ###
bins	200
bin_size 100000

### Iterations for EM - Optional (default is 100) ###
iterations 200

### Iterations for EM - Optional (default is 100) ###
#online 100000

### Initial lambda value for EM in units of mrad^2/cm - Optional (default is 0.1) ###
### If STD option is chosen, intial lambda is based off of poca results ###
lambda 0.1

### Milliradians option uses milliradians instead of radians in EM calculations ###
#milliradians
```

```
### Nominal option lets user set the nomimal momentum used in EM (default is 3) ###
#nominal 3

### Weighted EM - Optional selection will turn on weighting voxels in EM based on POCA (default
is turned off) ###
### Option 1 is pure POCA weight; i.e. voxel where POCA is found is weighted 1, all others 0###
### Option 2 is linear POCA weight; weight is determined by the formula (n - |c - p|) /n, n = voxels
before or after poca voxel, c = current voxel, p = poca voxel ###
#weight 2

### Output file - Optional ###
### Contains all information from EM preprocessing ###
#output pleasework.txt

### Standard deviation - Optional (default is off) - Can provide filename to output to separate file
(Used as intial value for EM lambda regardless) ###
#std test.std

### Angle distribution - Optional (default is off) - Can provide filename to output to separate file
###
#dist EMDIST.txt

### POCA - Optional (default is off) - Can provide filename to output to separate file ###
#poca VertSmallNew.txt

### Precise L & T - Use these options to estimate the L and T for every voxel rather than for every
event ###
precise_l
precise_t

#### Units - the metric units of length to run EM in relative to mm (i.e. 1m is 1000mm, so for meter
units use 1000 as the parameter) - default value is 10 (i.e. cm)
#units 10
```

MTS Suite

```
//mts.h
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include <time.h>

//define various program constants
#define DELIMS " \n\t"
#define ERR_PREC 0.000001

//define various maximum constants
#define MAX_COMMAND   100
#define MAX_FILENAME  200
#define MAX_LINE      1000

//define constants used for validation
#define CONTINUE     1
#define END          0
#define PARALLEL    -1

//define file pointer constants
#define FP_IN           0
#define FP_OUT          1
#define FP_OP_OUT       2
#define FP_STD_OUT      3
#define FP_DIST_OUT     4
#define FP_POCA_OUT     5
#define FP_OUT_AVG      6
#define FP_OUT_MED      7
#define MAX_FILEPOINTERS 8

//define constants for error return values
```

```
#define ERR_CODE_COMMAND_LINE       1
#define ERR_CODE_EMPTY_FILE         2
#define ERR_CODE_INVALID_COMMAND    3
#define ERR_CODE_INVALID_FILE       4
#define ERR_CODE_INVALID_INPUT      5
#define ERR_CODE_MEM                6
#define ERR_CODE_NO_FILE            7
#define ERR_CODE_OPTIONAL_REQUIRED  8
#define ERR_CODE_UNIX               9
#define ERR_CODE_UNUSED_FILE        10

//define constant for where to print error messages
#define ERR_OUT stderr

//define array indexes for various command line option flags
#define PARAM_DETAILS       0
#define PARAM_DEPENDENT     1
#define PARAM_DIST          2
#define PARAM_EM_AVERAGE    3
#define PARAM_EM_MEDIAN     4
#define PARAM_EM_3D         5
#define PARAM_EM_WEIGHTED   6
#define PARAM_END_OF_PATH   7
#define PARAM_HIT_TARGET    8
#define PARAM_INDEPENDENT   9
#define PARAM_ITERATIONS    10
#define PARAM_INIT_LAMBDA   11
#define PARAM_PRECISE_L     12
#define PARAM_PRECISE_T     13
#define PARAM_STD           14
#define PARAM_X_LENGTH      15
#define PARAM_Y_LENGTH      16
#define PARAM_Z_LENGTH      17
#define PARAM_X_VOXEL_SIZE  18
```

```
#define PARAM_Y_VOXEL_SIZE  19
#define PARAM_Z_VOXEL_SIZE  20
#define PARAM_X_VOXEL_TOTAL 21
#define PARAM_Y_VOXEL_TOTAL 22
#define PARAM_Z_VOXEL_TOTAL 23
#define PARAM_ALL_VOXELS    24
#define PARAM_X_MIN         25
#define PARAM_Y_MIN         26
#define PARAM_Z_MIN         27
#define PARAM_X_MAX         28
#define PARAM_Y_MAX         29
#define PARAM_Z_MAX         30
#define PARAM_EVENTS        31
#define PARAM_INC_DECT      32
#define PARAM_OUT_DECT      33
#define PARAM_MOMENTUM      34
#define PARAM_LINE          35
#define PARAM_HITS          36
#define PARAM_PARALLEL      37
#define PARAM_SCAT_ANG      38
#define PARAM_L             39
#define PARAM_EM            40
#define PARAM_IN_VOLUME     41
#define PARAM_CUR_EVENT     42
#define PARAM_CONTINUE      43
#define PARAM_NOM_MOMENTUM  44
#define PARAM_MILLIRADIANS  45
#define PARAM_MIN_MOMENTUM  46
#define PARAM_CUTOFF_ANGLE  47
#define PARAM_OP_OUT        48
#define PARAM_POCA          49
#define PARAM_DOCA          50
#define PARAM_DTX           51
#define PARAM_DTY           52
```

```
#define PARAM_DX            53
#define PARAM_DY            54
#define PARAM_OUT           55
#define PARAM_C_PRINT       56
#define PARAM_EM_BINS       57
#define PARAM_EM_BIN_SIZE   58
#define PARAM_EM_ONLINE     59
#define PARAM_UNITS_LENGTH  60
#define PARAM_MOM_CUT       61
#define PARAM_MOM_HIGH_CUT  62
#define PARAM_MOM_LOW_CUT   63
#define PARAM_PREV_VOXEL    64
#define MAX_PARAMS          65

//define constants for command line options
#define LONG_BINS         "bins"
#define LONG_BIN_SIZE     "bin_size"
#define LONG_C_PRINT      "c_print"
#define LONG_CUTOFF       "cutoff"
#define LONG_DETAILS      "details"
#define LONG_DIST         "dist"
#define LONG_EM           "em"
#define LONG_EM_AVERAGE   "average"
#define LONG_EM_MEDIAN    "median"
#define LONG_EM_3D        "3D"
#define LONG_EM_WEIGHTED  "weight"
#define LONG_HELP         "help"
#define LONG_INPUT        "input"
#define LONG_ITERATIONS   "iterations"
#define LONG_LAMBDA       "lambda"
#define LONG_MILLIRADIANS "milliradians"
#define LONG_MOM_HIGH_CUT "mom_cut_high"
#define LONG_MOM_LOW_CUT  "mom_cut_low"
#define LONG_NOMINAL      "nominal"
```

```
#define LONG_ONLINE       "online"
#define LONG_OUTPUT       "output"
#define LONG_POCA         "poca"
#define LONG_PRECISE_L    "precise_l"
#define LONG_PRECISE_T    "precise_t"
#define LONG_STD          "std"
#define LONG_UNITS_LENGTH "units"
#define LONG_X            "x_voxel_size"
#define LONG_Y            "y_voxel_size"
#define LONG_Z            "z_voxel_size"

#define SHORT_BINS         'B'
#define SHORT_BIN_SIZE     'b'
#define SHORT_C_PRINT      'C'
#define SHORT_CUTOFF       'c'
#define SHORT_DETAILS      'd'
#define SHORT_DIST         'D'
#define SHORT_EM           'e'
#define SHORT_EM_WEIGHTED  'w'
#define SHORT_HELP         'h'
#define SHORT_INPUT        'i'
#define SHORT_ITERATIONS   'I'
#define SHORT_LAMBDA       'l'
#define SHORT_MILLIRADIANS 'm'
#define SHORT_MOM_HIGH_CUT 'K'
#define SHORT_MOM_LOW_CUT  'k'
#define SHORT_NOMINAL      'n'
#define SHORT_ONLINE       'O'
#define SHORT_OUTPUT       'o'
#define SHORT_POCA         'p'
#define SHORT_PRECISE_L    'L'
#define SHORT_PRECISE_T    'T'
#define SHORT_STD          's'
#define SHORT_UNITS_LENGTH 'u'
```

```
#define SHORT_X           'x'
#define SHORT_Y           'y'
#define SHORT_Z           'z'

#define SHORT_OPTIONS  "B:b:c:dD::e::whi:I:k:K:l:Lmn:O:o::p::s::Tu:x:y:z:"

#define SAMPLE_VOXELS {24603, 23384, 23385, 23386, 23413, 23414, 23415, 24584, 24585,
24613, 24614, 24615, 24626, -1}
//#define SAMPLE_VOXELS {14374, 17994, 20638, 22735,  -1}
#define NUMOF_SAMP_VOX 13

//cosmetic constants
#define BANNER "======================"

struct Point {
  double x, y, z;
};
```

```
//driver.c
#include "mts.h"
#include "mtserr.h"
#include "preprocessing.h"
#include "em.h"
#include <getopt.h>

void set_default_parameters(double**);
int get_opts(int, double**, char**, FILE**);
int getopt_file(FILE*, struct option*);
FILE* fopen_ext(char*, char*, char*);
char* create_file_ext(char*, int, int, double**);

int main (int argc, char **argv) {

   int i, errCode;
   double *params[MAX_PARAMS], *lambda, *lambdaMed, *M;
   FILE *filepointers[MAX_FILEPOINTERS];
   struct muon* head;

   for (i=0;i<MAX_PARAMS;i++) if ((params[i] = (double*) malloc(sizeof(double)))==NULL)
return memError();
   for (i=0;i<MAX_FILEPOINTERS;i++) if ((filepointers[i] = (FILE*)
malloc(sizeof(FILE)))==NULL) return memError();
   if ((head=(struct muon*) malloc(sizeof(struct muon)))==NULL) return memError();

   set_default_parameters(params);

   if (get_opts(argc, params, argv, filepointers)!=CONTINUE) return
ERR_CODE_COMMAND_LINE;
   free(optarg);

   preprocessing(lambda, M, head, params, filepointers);
   if (*params[PARAM_EM] && !*params[PARAM_EM_ONLINE]) em(lambda, lambdaMed,
```

```
head, params, filepointers);

   for (i=0;i<MAX_PARAMS;i++) free(params[i]);
   for (i=0;i<MAX_FILEPOINTERS;i++) free(filepointers[i]);

   fprintf(stderr, "\n");

   return errCode;
}

void set_default_parameters (double** params) {

   int i;

   //all paramaters are intially zero unless otherwise set in this modle
   for (i=0;i<MAX_PARAMS;i++) *params[i] = 0;

   *params[PARAM_X_LENGTH] = 4000;
   *params[PARAM_Y_LENGTH] = 4000;
   *params[PARAM_Z_LENGTH] = 3000;

   *params[PARAM_X_VOXEL_SIZE] = 100;
   *params[PARAM_Y_VOXEL_SIZE] = 100;
   *params[PARAM_Z_VOXEL_SIZE] = 100;

   *params[PARAM_X_VOXEL_TOTAL] = *params[PARAM_X_LENGTH] /
*params[PARAM_X_VOXEL_SIZE];
   *params[PARAM_Y_VOXEL_TOTAL] = *params[PARAM_Y_LENGTH] /
*params[PARAM_Y_VOXEL_SIZE];
   *params[PARAM_Z_VOXEL_TOTAL] = *params[PARAM_Z_LENGTH] /
*params[PARAM_Z_VOXEL_SIZE];
   *params[PARAM_ALL_VOXELS]    = *params[PARAM_X_VOXEL_TOTAL] *
*params[PARAM_Y_VOXEL_TOTAL] * *params[PARAM_Z_VOXEL_TOTAL];
```

```
  *params[PARAM_X_MIN]        = *params[PARAM_X_LENGTH] / -2;
  *params[PARAM_Y_MIN]        = *params[PARAM_Y_LENGTH] / -2;
  *params[PARAM_Z_MIN]        = *params[PARAM_Z_LENGTH] / -2;

  *params[PARAM_X_MAX]        = *params[PARAM_X_LENGTH] / 2;
  *params[PARAM_Y_MAX]        = *params[PARAM_Y_LENGTH] / 2;
  *params[PARAM_Z_MAX]        = *params[PARAM_Z_LENGTH] / 2;

  *params[PARAM_CONTINUE] = 1;

  *params[PARAM_ITERATIONS]  = 100;
  *params[PARAM_INIT_LAMBDA] = 0.1;

  *params[PARAM_NOM_MOMENTUM] = 3;
  *params[PARAM_MIN_MOMENTUM] = 1;
  *params[PARAM_MILLIRADIANS] = 1;

  *params[PARAM_EM] = 0;
  *params[PARAM_EM_BIN_SIZE] = 10000;
  *params[PARAM_EM_BINS] = 100;

  *params[PARAM_UNITS_LENGTH] = 10;

  return;
}

int get_opts (int argc, double** params, char** argv, FILE** fps) {

  char fnOut[MAX_FILENAME], extAvg[MAX_FILENAME], extMed[MAX_FILENAME];
  int option, argvIndex=0, i=0, places=0, zeros;
  FILE* config = NULL;

  static struct option long_options[] = {
    {LONG_BINS,        required_argument, NULL,        SHORT_BINS},
```

```
    {LONG_BIN_SIZE,     required_argument, NULL,     SHORT_BIN_SIZE},
    {LONG_C_PRINT,      required_argument, NULL,      SHORT_C_PRINT},
    {LONG_CUTOFF,       required_argument, NULL,       SHORT_CUTOFF},
    {LONG_DETAILS,      no_argument,       NULL,      SHORT_DETAILS},
    {LONG_DIST,         required_argument, NULL,         SHORT_DIST},
    {LONG_EM,           optional_argument, NULL,           SHORT_EM},
    {LONG_EM_WEIGHTED,  no_argument,       NULL,  SHORT_EM_WEIGHTED},
    {LONG_HELP,         no_argument,       NULL,         SHORT_HELP},
    {LONG_INPUT,        required_argument, NULL,        SHORT_INPUT},
    {LONG_ITERATIONS,   required_argument, NULL,   SHORT_ITERATIONS},
    {LONG_LAMBDA,       required_argument, NULL,       SHORT_LAMBDA},
    {LONG_MILLIRADIANS, no_argument,       NULL, SHORT_MILLIRADIANS},
    {LONG_MOM_HIGH_CUT, required_argument, NULL, SHORT_MOM_HIGH_CUT},
    {LONG_MOM_LOW_CUT,  required_argument, NULL,  SHORT_MOM_LOW_CUT},
    {LONG_NOMINAL,      required_argument, NULL,      SHORT_NOMINAL},
    {LONG_ONLINE,       required_argument, NULL,       SHORT_ONLINE},
    {LONG_OUTPUT,       required_argument, NULL,       SHORT_OUTPUT},
    {LONG_POCA,         required_argument, NULL,         SHORT_POCA},
    {LONG_PRECISE_L,    no_argument,       NULL,    SHORT_PRECISE_L},
    {LONG_PRECISE_T,    no_argument,       NULL,    SHORT_PRECISE_T},
    {LONG_STD,          optional_argument, NULL,          SHORT_STD},
    {LONG_UNITS_LENGTH, required_argument, NULL, SHORT_UNITS_LENGTH},
    {LONG_X,            required_argument, NULL,            SHORT_X},
    {LONG_Y,            required_argument, NULL,            SHORT_Y},
    {LONG_Z,            required_argument, NULL,            SHORT_Z},
    {0, 0, 0, 0}
  };

  if (argc<=1) return commandError(SHORT_HELP, SHORT_HELP);
  if (argv[1][0]!='-') if ((config=fopen(argv[1], "r"))==NULL) return
commandError(SHORT_HELP, SHORT_HELP);

  while (1) {
```

```
    if (config==NULL) option = getopt_long (argc, argv, SHORT_OPTIONS, long_options,
&argvIndex);
    else option = getopt_file (config, long_options);
    if (option == -1) break;

    switch (option) {
      case SHORT_BINS:
        *params[PARAM_EM_BINS] = atof(optarg);
        break;
      case SHORT_BIN_SIZE:
        *params[PARAM_EM_BIN_SIZE] = atof(optarg);
        break;
      case SHORT_C_PRINT:
        *params[PARAM_C_PRINT] = 1;
        break;
      case SHORT_CUTOFF:
        *params[PARAM_CUTOFF_ANGLE] = atof(optarg);
        break;
      case SHORT_DETAILS:
        *params[PARAM_DETAILS] = 1;
        break;
      case SHORT_DIST:
        if (optarg!=NULL) fps[FP_DIST_OUT] = fopen(optarg, "w");
        else              fps[FP_DIST_OUT] = fopen("default.dist", "w");
        *params[PARAM_OUT] = *params[PARAM_DIST] = 1;
        break;
      case SHORT_EM:
        *params[PARAM_EM] = 2;
        if (optarg==NULL) *params[PARAM_EM_AVERAGE] = 1;
        else if (strcmp(optarg, LONG_EM_AVERAGE)==0) *params[PARAM_EM_AVERAGE] =
1;
        else if (strcmp(optarg, LONG_EM_MEDIAN)==0) *params[PARAM_EM_MEDIAN] = 1;
        else if (strcmp(optarg, LONG_EM_3D)==0) *params[PARAM_EM_3D] = 1;
        break;
```

```
case SHORT_EM_WEIGHTED:
  *params[PARAM_EM_WEIGHTED] = 1;
  break;
case SHORT_HELP:
  return (commandError(option, option));
  break;
case SHORT_INPUT:
  if ((fps[FP_IN] = fopen(optarg, "r"))==NULL) return fileError(optarg);
  strcpy(fnOut, optarg);
  break;
case SHORT_ITERATIONS:
  *params[PARAM_ITERATIONS] = atof(optarg);
  break;
case SHORT_LAMBDA:
  *params[PARAM_INIT_LAMBDA] = atof(optarg);
  for (i=0; optarg[i]!='.' && optarg[i]!='\0'; i++);
  for (i=i+1, places=0, zeros=0; optarg[i]=='0'; places++, zeros++, i++);
  for (;optarg[i]!='\0'; places++, i++);
  break;
case SHORT_MILLIRADIANS:
  *params[PARAM_MILLIRADIANS] = 1000;
  break;
case SHORT_MOM_HIGH_CUT:
  *params[PARAM_MOM_HIGH_CUT] = atof(optarg);
  *params[PARAM_MOM_CUT] = 1;
  break;
case SHORT_MOM_LOW_CUT:
  *params[PARAM_MOM_LOW_CUT] = atof(optarg);
  *params[PARAM_MOM_CUT] = 1;
  break;
case SHORT_NOMINAL:
  *params[PARAM_NOM_MOMENTUM] = atof(optarg);
  break;
case SHORT_ONLINE:
```

```
    *params[PARAM_EM_ONLINE] = atof(optarg);
    break;
  case SHORT_OUTPUT:
    if (optarg!=NULL) fps[FP_OP_OUT] = fopen(optarg, "w");
    else              fps[FP_OP_OUT] = fopen("default.opout", "w");
    *params[PARAM_OUT] = *params[PARAM_OP_OUT] = 1;
    break;
  case SHORT_POCA:
    if (optarg!=NULL) fps[FP_POCA_OUT] = fopen(optarg, "w");
    else              fps[FP_POCA_OUT] = fopen("default.poca", "w");
    *params[PARAM_OUT] = *params[PARAM_POCA] = 1;
    break;
  case SHORT_PRECISE_L:
    *params[PARAM_PRECISE_L] = 1;
    break;
  case SHORT_PRECISE_T:
    *params[PARAM_PRECISE_T] = 1;
    break;
  case SHORT_STD:
    if (optarg!=NULL) fps[FP_STD_OUT] = fopen(optarg, "w");
    else fps[FP_STD_OUT] = NULL;
    *params[PARAM_STD] = 1;
    break;
  case SHORT_UNITS_LENGTH:
    *params[PARAM_UNITS_LENGTH] = atof(optarg);
    break;
  case SHORT_X:
    *params[PARAM_X_VOXEL_SIZE] = atof(optarg);
    break;
  case SHORT_Y:
    *params[PARAM_Y_VOXEL_SIZE] = atof(optarg);
    break;
  case SHORT_Z:
    *params[PARAM_Z_VOXEL_SIZE] = atof(optarg);
```

```
          break;
        case '?':
          return (commandError(optopt, optopt));
          break;
        default:
          printf ("\n\nIf you are seeing this then quantum mechanics is for real: %c", option);
      }
      free(optarg);
  }
  if (config!=NULL) fclose(config);

  if (*params[PARAM_EM_3D]) *params[PARAM_EM] = 1;

  if (*params[PARAM_EM_AVERAGE]) {
    sprintf(extAvg, "avg");
    if ((fps[FP_OUT_AVG] = fopen_ext(fnOut, create_file_ext(extAvg, places, zeros, params),
"w"))==NULL) return fileError(fnOut);
  }
  if (*params[PARAM_EM_MEDIAN]) {
    sprintf(extMed, "med%dbins%dsize", (int) *params[PARAM_EM_BINS], (int)
*params[PARAM_EM_BIN_SIZE]);
    if ((fps[FP_OUT_MED] = fopen_ext(fnOut, create_file_ext(extMed, places, zeros, params),
"w"))==NULL) return fileError(fnOut);
  }

  if (*params[PARAM_MILLIRADIANS]==1)  *params[PARAM_INIT_LAMBDA] =
*params[PARAM_INIT_LAMBDA] / 1000000;
  if (*params[PARAM_UNITS_LENGTH]!=10) *params[PARAM_INIT_LAMBDA] =
*params[PARAM_INIT_LAMBDA] / (10 / *params[PARAM_UNITS_LENGTH]);

  return CONTINUE;
}
```

```
int getopt_file(FILE* config, struct option* options) {

  int i;
  char line[MAX_LINE], *token;

  optarg = (char*) malloc(MAX_LINE);
  while (1) {
    if (fgets(line, MAX_LINE-1, config)==NULL) return -1;

    if ((token = strtok(line, DELIMS))==NULL) continue;
    if (token[0]=='#') continue;

    for (i=0; options[i].val!=0; i++) {
      if (strcmp(token, options[i].name)==0) {
        if ((token = strtok(NULL, DELIMS))!=NULL) strcpy(optarg, token);
        else {
          free(optarg);
          optarg=NULL;
        }
        return options[i].val;
      }
    }
    return '?';
  }
}

FILE* fopen_ext(char* fn, char* ext, char* param) {

  char newFn[200];
  int i, j;

  strcpy(newFn, fn);
  j = 0;
  i = strcspn(fn, ".");
```

```
  while (ext[j]!='\0') newFn[++i] = ext[j++];
  newFn[++i]='\0';

  return fopen(newFn, param);

}

char* create_file_ext(char* ext, int places, int zeros, double** params) {

  int i, after_zero = (int) (ceil((((double) (*params[PARAM_INIT_LAMBDA] - ((int)
*params[PARAM_INIT_LAMBDA]))) * pow(10, ((double) places)))));

  if (*params[PARAM_EM_3D])            sprintf(ext, "%s-3D", ext);
  if (*params[PARAM_X_VOXEL_SIZE])      sprintf(ext, "%s-vox%dcm", ext, (int)
(*params[PARAM_X_VOXEL_SIZE]/10));
  if (*params[PARAM_ITERATIONS])       sprintf(ext, "%s-itr%d", ext, (int)
*params[PARAM_ITERATIONS]);

  if (*params[PARAM_INIT_LAMBDA]) {
    sprintf(ext, "%s-lam%i.", ext, (int) *params[PARAM_INIT_LAMBDA]);
    for (i=0; i<zeros; i++) sprintf(ext, "%s0", ext);
    sprintf(ext, "%s%i", ext, after_zero);
  }

  if (*params[PARAM_CUTOFF_ANGLE])      sprintf(ext, "%s-angcut%d", ext, (int)
*params[PARAM_CUTOFF_ANGLE]);
  if (*params[PARAM_EM_WEIGHTED])       sprintf(ext, "%s-weight%d", ext, (int)
*params[PARAM_EM_WEIGHTED]);
  if (*params[PARAM_EM_ONLINE])         sprintf(ext, "%s-inc%d", ext, (int)
*params[PARAM_EM_ONLINE]);
  if (*params[PARAM_PRECISE_L])         sprintf(ext, "%s-pL", ext);
  if (*params[PARAM_PRECISE_T])         sprintf(ext, "%s-pT", ext);
  if (*params[PARAM_MOM_CUT] = 1)       sprintf(ext, "%s-momcutHi%dLo%d", ext, (int)
```

```
*params[PARAM_MOM_HIGH_CUT], (int) *params[PARAM_MOM_LOW_CUT]);
  if (*params[PARAM_UNITS_LENGTH])       sprintf(ext, "%s-units%fmm", ext,
*params[PARAM_UNITS_LENGTH]);
  if (*params[PARAM_MILLIRADIANS]==1000) sprintf(ext, "%s-mrad", ext);

  return ext;
}
```

```
//preprocessing.h
#define TRACK_PUSH 0.00000001
#define SMALL_NUM  0.1
#define INPUT_FILE "geant_input_file"

int preprocessing(double*, double*, struct muon*, double**, FILE**);
int header(double*, double*, double**, FILE**);
int em_data(struct Line*, struct Line*, struct muon*, double**, FILE**);
double pocaLtoL(struct Line*, struct Line*, struct Point*, double**);
void travel(struct Point*, struct Point*, int, double);
double in_volume(struct Point*, double**);
struct voxel* track(struct Point*, struct Point*, struct Point*, struct muon*, struct voxel*, double*,
double**, FILE**);
```

```
//preprocess.c
#include "mts.h"
#include "preprocessing.h"
#include "mtserr.h"
#include "mtsio.h"
#include "vecfunc.h"

int preprocessing(double* lambda, double* M, struct muon* head, double** params, FILE** fps) {

  fprintf(ERR_OUT, "%s POCA RECONSTRUCTION %s\n\n", BANNER, BANNER);

  int i, j, voxel, xVox, yVox, zVox, errCode = CONTINUE;
  double *voxel_std, *voxel_avg, *voxel_n, *doca, *lambdaLocal, *lambdaMedLocal;

  //pointers to incoming and outgoing muon tracks
  struct Line *muonInc, *muonOut;

  //create pointers to detector points on incoming/outgoing muon tracks
  struct Point *incPoints[MAX_DETECTORS], *outPoints[MAX_DETECTORS];
  struct Point *scatPt, *tempP;

  struct muon *curMuon;
  struct voxel *trackHead, *dummy = (struct voxel*) malloc(sizeof(struct voxel));

  //set aside memory for the different structs
  if ((muonInc = (struct Line*) malloc(sizeof(struct Line)))==NULL) return memError();
  if (((*muonInc).P1 = (struct Point*) malloc(sizeof(struct Point)))==NULL) return memError();
  if (((*muonInc).P2 = (struct Point*) malloc(sizeof(struct Point)))==NULL) return memError();
  if ((muonOut = (struct Line*) malloc(sizeof(struct Line)))==NULL) return memError();
  if (((*muonOut).P1 = (struct Point*) malloc(sizeof(struct Point)))==NULL) return memError();
  if (((*muonOut).P2 = (struct Point*) malloc(sizeof(struct Point)))==NULL) return memError();
  if ((scatPt = (struct Point*) malloc(sizeof(struct Point)))==NULL) return memError();
  for (i = 0; i < MAX_DETECTORS; i++) {
    if ((incPoints[i] = (struct Point*) malloc(sizeof(struct Point)))==NULL) return memError();
```

```
    if ((outPoints[i] = (struct Point*) malloc(sizeof(struct Point)))==NULL) return memError();
  }

  if (header(lambda, M, params, fps) != CONTINUE) return 0;

  if ((doca = (double*) malloc(sizeof(double) * *params[PARAM_EVENTS]/5))==NULL) return
memError();

  if (*params[PARAM_STD]) {
    if ((voxel_std = (double*) malloc(sizeof(double) *
*params[PARAM_ALL_VOXELS]))==NULL) return memError();
    if ((voxel_avg = (double*) malloc(sizeof(double) *
*params[PARAM_ALL_VOXELS]))==NULL) return memError();
    if ((voxel_n   = (double*) malloc(sizeof(double) *
*params[PARAM_ALL_VOXELS]))==NULL) return memError();

    memset(voxel_std, '\0', sizeof(double) * *params[PARAM_ALL_VOXELS]);
    memset(voxel_avg, '\0', sizeof(double) * *params[PARAM_ALL_VOXELS]);
    memset(voxel_n, '\0', sizeof(double) * *params[PARAM_ALL_VOXELS]);
  }

  curMuon = head;
  while(errCode!=END) {

    //get muon tracks from either input file or manual input
    errCode = get_geant_input(incPoints, outPoints, params, fps);
    if (*params[PARAM_MOM_CUT]) {
      if ((*params[PARAM_MOMENTUM] > *params[PARAM_MOM_HIGH_CUT]) &&
(*params[PARAM_MOM_HIGH_CUT])) continue;
      if ((*params[PARAM_MOMENTUM] < *params[PARAM_MOM_LOW_CUT]) &&
(*params[PARAM_MOM_LOW_CUT])) continue;
    }

    if (errCode==END) {
```

```
  if (*params[PARAM_STD]==1) {
    (*params[PARAM_STD])++;
    *params[PARAM_CUR_EVENT]=0;
    rewind(fps[FP_IN]);
    for (i=0;i<*params[PARAM_ALL_VOXELS];i++) voxel_avg[i] = voxel_avg[i] / voxel_n[i];
    errCode=CONTINUE;
    continue;
  } else break;
}
(*params[PARAM_CUR_EVENT])++;

if (errCode==ERR_CODE_INVALID_INPUT) continue;

vec_fit(incPoints, muonInc, (int) *params[PARAM_INC_DECT], FIT_X);
vec_fit(incPoints, muonInc, (int) *params[PARAM_INC_DECT], FIT_Y);
vec_fit(outPoints, muonOut, (int) *params[PARAM_OUT_DECT], FIT_X);
vec_fit(outPoints, muonOut, (int) *params[PARAM_OUT_DECT], FIT_Y);

//find point of closest approach between incoming/outgoing muon tracks and return doca
*params[PARAM_DOCA] = pocaLtoL(muonInc, muonOut, scatPt, params);

//check if tracks were parallel and if the scatter point is inside the detector volume and set flag to
0 if not
if (*params[PARAM_DOCA]==PARALLEL)*params[PARAM_CONTINUE] = 0;
if (!(in_volume(scatPt, params))) *params[PARAM_CONTINUE] = 0;

*params[PARAM_SCAT_ANG] = vec_angle(muonInc, muonOut, ALL_COMPONENTS);

if (*params[PARAM_STD] && *params[PARAM_CONTINUE]) {

  xVox = (int) floor((((*scatPt).x + *params[PARAM_X_MAX]) /
*params[PARAM_X_VOXEL_SIZE]));
  yVox = (int) floor((((*scatPt).y + *params[PARAM_Y_MAX]) /
*params[PARAM_Y_VOXEL_SIZE]));
```

```
        zVox = (int) floor((((*scatPt).z + *params[PARAM_Z_MAX]) /
*params[PARAM_Z_VOXEL_SIZE]));

        //voxel number determined in z direction first, then y, then x
        voxel = (xVox * *params[PARAM_Y_VOXEL_TOTAL] *
*params[PARAM_Z_VOXEL_TOTAL]) + (yVox * *params[PARAM_Z_VOXEL_TOTAL]) +
zVox;

        if (*params[PARAM_STD]==1) {
          voxel_avg[voxel] = voxel_avg[voxel] + *params[PARAM_SCAT_ANG];
          voxel_n[voxel]++;
        }

        if (*params[PARAM_STD]==2) voxel_std[voxel] = voxel_std[voxel] +
pow((*params[PARAM_SCAT_ANG] - voxel_avg[voxel]), 2);

      }

      //if em analysis is to be done, print the appropriate data to file or store it in the appropriate data
structures
      if (((*curMuon).nextMuon=(struct muon*) malloc(sizeof(struct muon)))==NULL) return
memError();
      curMuon = (*curMuon).nextMuon;
      if ((errCode = em_data(muonInc, muonOut, curMuon, params, fps))!=CONTINUE) return
errCode;

      if ((*curMuon).event==7699) fprintf(stderr, "angle %f dx %f\n", (*curMuon).dX,
(*curMuon).dtX);

      if (*params[PARAM_OUT]) write_optional((*muonInc).P2, scatPt, (*muonOut).P1, params,
fps);

      if (((*curMuon).muonTrack=(struct voxel*) malloc(sizeof(struct voxel)))==NULL) return
memError();
```

```
    //the following block of code takes care of track analysis; if the scatter point was valid then the
track is calculate along the POCA
    //path, if not then the path between the entering and exiting tracks is used
    trackHead = dummy;
    (*trackHead).nextVoxel = (*curMuon).muonTrack;
    if (*params[PARAM_CONTINUE]) {
      if ((trackHead = track((*muonInc).P2, scatPt, (*muonOut).P1, curMuon, trackHead, M,
params, fps))==NULL) return memError();
      if ((((*trackHead).nextVoxel=(struct voxel*) malloc(sizeof(struct voxel)))==NULL) return
memError();
      if ((trackHead = track(scatPt, (*muonOut).P1, NULL, curMuon, trackHead, M, params,
fps))==NULL) return memError();
    } else {
      if ((trackHead = track((*muonInc).P2, (*muonOut).P1, NULL, curMuon, trackHead, M,
params, fps))==NULL) return memError();
    }

    if (*params[PARAM_EM_ONLINE]) {
      if (fmod(*params[PARAM_CUR_EVENT], *params[PARAM_EM_ONLINE])==0)
em(lambdaLocal, lambdaMedLocal, head, params, fps);
    }

    *params[PARAM_PREV_VOXEL] = -1;
    *params[PARAM_CONTINUE]=1;
    //break;
  }
  (*curMuon).nextMuon = NULL;

  if (*params[PARAM_EM_ONLINE]) {
    if (fmod(*params[PARAM_CUR_EVENT], *params[PARAM_EM_ONLINE])!=0)
em(lambdaLocal, lambdaMedLocal, head, params, fps);
    free(lambdaLocal);
    free(lambdaMedLocal);
```

```
  }

  fprintf(stderr, "\n\nPARALLEL PARAM: %.10f\n\n", SMALL_NUM);
  fprintf(stderr, "PARALLEL TRACKS: %f\n", *params[PARAM_PARALLEL]);
  fprintf(stderr, "TOTAL    TRACKS: %f\n", *params[PARAM_CUR_EVENT]);

  double max = 0;
  if (*params[PARAM_STD]) {
    for (i=0;i<*params[PARAM_ALL_VOXELS];i++) {
      voxel_std[i] = pow((voxel_std[i] / voxel_n[i]), 0.5);
      if (voxel_std[i]>max) max=voxel_std[i];
      //if (*params[PARAM_EM]) lambda[i] = voxel_std[i];
    }
    fprintf(stderr, "max = %f\n\n", max);
    if (fps[FP_STD_OUT]!=NULL) write_lambda(voxel_std, voxel_n, NULL, params,
fps[FP_STD_OUT]);
    free(voxel_avg);
    free(voxel_n);
    free(voxel_std);
  }

  free(dummy);
  free(doca);
  free((*muonInc).P1);
  free((*muonInc).P2);
  free((*muonOut).P1);
  free((*muonOut).P2);
  free(muonInc);
  free(muonOut);

  for (i = 0; i <MAX_DETECTORS; i++) {
    free(incPoints[i]);
    free(outPoints[i]);
  }
```

```
  return errCode;
}

//header processes the first 2 lines from the input file which containts the number of events in
//the run and the lenght of the detector volume in x, y and z and then determines other paramaters
//based on available information
int header(double* lambda, double* M, double** params, FILE** fps) {

  int i;
  char numEvents[20]; //string which contains number of *params[PARAM_EVENTS] in
simulation

  //gets the number of total *params[PARAM_EVENTS] from the second *params[PARAM_LINE]
in the input file
  if ((fgets(numEvents, 100, fps[FP_IN])==NULL)) return emptyError(INPUT_FILE);
  (*params[PARAM_LINE])++;

  if (numEvents[0]!=EVENTS_START) return formatError(INPUT_FILE, numEvents,
(*params[PARAM_LINE])++);
  if ((fgets(numEvents, 100, fps[FP_IN])==NULL)) return emptyError(INPUT_FILE);

  if ((*params[PARAM_EVENTS] = atof(strtok(numEvents, DELIMS))) == 0) return
formatError(INPUT_FILE, numEvents, (*params[PARAM_LINE])++);

  *params[PARAM_X_LENGTH] = atof(strtok(NULL, DELIMS));
  if (*params[PARAM_X_LENGTH]==0) return formatError(INPUT_FILE, numEvents,
(*params[PARAM_LINE])++);

  *params[PARAM_Y_LENGTH] = atof(strtok(NULL, DELIMS));
  if (*params[PARAM_Y_LENGTH]==0) return formatError(INPUT_FILE, numEvents,
(*params[PARAM_LINE])++);

  *params[PARAM_Z_LENGTH] = atof(strtok(NULL, DELIMS));
```

```
  if (*params[PARAM_Z_LENGTH]==0) return formatError(INPUT_FILE, numEvents,
(*params[PARAM_LINE])++);

  *params[PARAM_X_MAX] = *params[PARAM_X_LENGTH]/2;
  *params[PARAM_Y_MAX] = *params[PARAM_Y_LENGTH]/2;
  *params[PARAM_Z_MAX] = *params[PARAM_Z_LENGTH]/2;

  *params[PARAM_X_MIN] = *params[PARAM_X_LENGTH]/2 * -1;
  *params[PARAM_Y_MIN] = *params[PARAM_Y_LENGTH]/2 * -1;
  *params[PARAM_Z_MIN] = *params[PARAM_Z_LENGTH]/2 * -1;

  *params[PARAM_X_VOXEL_TOTAL] = *params[PARAM_X_LENGTH] /
*params[PARAM_X_VOXEL_SIZE];
  *params[PARAM_Y_VOXEL_TOTAL] = *params[PARAM_Y_LENGTH] /
*params[PARAM_Y_VOXEL_SIZE];
  *params[PARAM_Z_VOXEL_TOTAL] = *params[PARAM_Z_LENGTH] /
*params[PARAM_Z_VOXEL_SIZE];

  *params[PARAM_ALL_VOXELS] = *params[PARAM_X_VOXEL_TOTAL] *
*params[PARAM_Y_VOXEL_TOTAL] * *params[PARAM_Z_VOXEL_TOTAL];

  return CONTINUE;
}

// pocaLtoL():
//    Input:  two lines L1 and L2:
//    Return: the shortest distance between L1 and L2
double pocaLtoL(struct Line *L1, struct Line *L2, struct Point* scatPt, double** params) {

  double a, b, c, d, e, D, sc, tc, doca;

  struct Point *u, *v, *w, *dPVector, *scXu, *tcXv, *scXuMtcXv, *cp1, *cp2;
  if ((u        = (struct Point*) malloc(sizeof(struct Point)))==NULL) return memError();
  if ((v        = (struct Point*) malloc(sizeof(struct Point)))==NULL) return memError();
```

```
if ((w         = (struct Point*) malloc(sizeof(struct Point)))==NULL) return memError();
if ((dPVector  = (struct Point*) malloc(sizeof(struct Point)))==NULL) return memError();
if ((scXu      = (struct Point*) malloc(sizeof(struct Point)))==NULL) return memError();
if ((tcXv      = (struct Point*) malloc(sizeof(struct Point)))==NULL) return memError();
if ((scXuMtcXv = (struct Point*) malloc(sizeof(struct Point)))==NULL) return memError();
if ((cp1       = (struct Point*) malloc(sizeof(struct Point)))==NULL) return memError();
if ((cp2       = (struct Point*) malloc(sizeof(struct Point)))==NULL) return memError();

vec_sub((*L1).P2, (*L1).P1, u);
vec_sub((*L2).P2, (*L2).P1, v);
vec_sub((*L1).P1, (*L2).P1, w);

a = vec_dot(u, u);
b = vec_dot(u, v);
c = vec_dot(v, v);
d = vec_dot(u, w);
e = vec_dot(v, w);

D = (a)*(c) - (b)*(b);

// compute the line parameters of the two closest points
if (D < SMALL_NUM) {        // the lines are almost parallel

   (*params[PARAM_PARALLEL])++;

   (*scatPt).x = (*((*L1).P2)).x;
   (*scatPt).y = (*((*L1).P2)).y;
   (*scatPt).z = (*((*L1).P2)).z;

   return PARALLEL;
} else {

   sc = ((b)*(e) - (c)*(d)) / D;
   tc = ((a)*(e) - (b)*(d)) / D;
```

```
    }

    vec_mult(sc, u, scXu);
    vec_mult(tc, v, tcXv);

    //next three lines are for distance of closest approach
    vec_sub(scXu, tcXv, scXuMtcXv);
    vec_add(w, scXuMtcXv, dPVector);
    doca = vec_norm(dPVector);

    //next three lines are for point of closest approach
    vec_add((*L1).P1, scXu, cp1);
    vec_add((*L2).P1, tcXv, cp2);
    vec_mid(cp1, cp2, scatPt);

    free(cp2);
    free(cp1);
    free(scXuMtcXv);
    free(scXu);
    free(tcXv);
    free(dPVector);
    free(w);
    free(v);
    free(u);

    return doca;
}


double in_volume(struct Point* p, double** params) {

    if (((*p).x > *params[PARAM_X_MAX]) || ((*p).x < *params[PARAM_X_MIN])) return 0;
    if (((*p).y > *params[PARAM_Y_MAX]) || ((*p).y < *params[PARAM_Y_MIN])) return 0;
```

```
  if (((*p).z > *params[PARAM_Z_MAX]) || ((*p).z < *params[PARAM_Z_MIN])) return 0;

  return 1;
}


int em_data(struct Line* muonIn, struct Line* muonOut, struct muon* mu, double** params,
FILE** fps) {

  int i, j;
  double thetaX, thetaX0, thetaX1, thetaY, thetaY0, thetaY1, difX, difY, deltaX, deltaY, pr2, Lxy;

  struct Line *vertIn, *vertOut;
  struct Point *dvIn, *p1, *distance;

  if ((vertIn = (struct Line*) malloc(sizeof(struct Line)))==NULL) return memError();
  if (((*vertIn).P1 = (struct Point*) malloc(sizeof(struct Point)))==NULL) return memError();
  if (((*vertIn).P2 = (struct Point*) malloc(sizeof(struct Point)))==NULL) return memError();
  if ((vertOut = (struct Line*) malloc(sizeof(struct Line)))==NULL) return memError();
  if (((*vertOut).P1 = (struct Point*) malloc(sizeof(struct Point)))==NULL) return memError();
  if (((*vertOut).P2 = (struct Point*) malloc(sizeof(struct Point)))==NULL) return memError();

  if ((dvIn = (struct Point*) malloc(sizeof(struct Point)))==NULL) return memError();
  if ((p1 = (struct Point*) malloc(sizeof(struct Point)))==NULL) return memError();

  //dvIn is the position vector of the incoming muon track
  vec_sub((*muonIn).P2, (*muonIn).P1, dvIn);

  //setting up vertical vectors for the top and bottom outgoing tracks
  vec_copy(muonIn, vertIn);
  (*(*vertIn).P1).x = (*(*muonIn).P2).x;
  (*(*vertIn).P1).y = (*(*muonIn).P2).y;

  vec_copy(muonOut, vertOut);
```

```
(*(*vertOut).P2).x = (*(*muonOut).P1).x;
(*(*vertOut).P2).y = (*(*muonOut).P1).y;

thetaX0 = vec_angle(muonIn, vertIn, X_COMPONENT);
thetaX1 = vec_angle(muonOut, vertOut, X_COMPONENT);
if (*params[PARAM_EM_3D]) thetaX = *params[PARAM_SCAT_ANG];
else                      thetaX  = thetaX1 - thetaX0;

thetaY0 = vec_angle(muonIn, vertIn, Y_COMPONENT);
thetaY1 = vec_angle(muonOut, vertOut, Y_COMPONENT);
if (*params[PARAM_EM_3D]) thetaY = 0;
else                      thetaY  = thetaY1 - thetaY0;

Lxy = sqrt(1 + pow(tan(thetaX0), 2) + pow(tan(thetaY0), 2));
*params[PARAM_L] = (*params[PARAM_X_VOXEL_SIZE] * Lxy) / 10;

(*p1).x = (*(*muonIn).P2).x;
(*p1).y = (*(*muonIn).P2).y;
(*p1).z = (*(*muonIn).P2).z;
travel(p1, dvIn, Z_COMPONENT, (*(*muonOut).P1).z);

difX = ((*(*muonOut).P1).x - (*p1).x);
difY = ((*(*muonOut).P1).y - (*p1).y);
deltaX = difX * cos(thetaX0) * Lxy * (cos(thetaX + thetaX0) / cos(thetaX));
deltaY = difY * cos(thetaY0) * Lxy * (cos(thetaY + thetaY0) / cos(thetaY));


(*mu).event = (int) *params[PARAM_CUR_EVENT];
if (*params[PARAM_DOCA]==PARALLEL || isnan(thetaX) || isnan(thetaY)) {
  (*mu).dtX   = 0;
  (*mu).dtY   = 0;
  (*mu).dX    = 0;
  (*mu).dY    = 0;
} else {
```

```
    (*mu).dtX   = fabs(thetaX * *params[PARAM_MILLIRADIANS]);
    (*mu).dtY   = fabs(thetaY * *params[PARAM_MILLIRADIANS]);
    (*mu).dX    = fabs(deltaX / *params[PARAM_UNITS_LENGTH]);
    (*mu).dY    = fabs(deltaY / *params[PARAM_UNITS_LENGTH]);
  }
  (*mu).pr2   = pow(*params[PARAM_NOM_MOMENTUM] /
*params[PARAM_MOMENTUM], 2);

  *params[PARAM_DTX] = (*mu).dtX;
  *params[PARAM_DTY] = (*mu).dtY;
  *params[PARAM_DX]  = (*mu).dX;
  *params[PARAM_DY]  = (*mu).dY;

  (*mu).a   = 0;
  (*mu).b   = -1;

  free(p1);
  free(dvIn);
  free((*vertIn).P1);
  free((*vertIn).P2);
  free((*vertOut).P1);
  free((*vertOut).P2);
  free(vertIn);
  free(vertOut);

  return CONTINUE;
}

//travel moves a point a long a vector
// p is the Point to move
// V is the position vector
// xyz is what component is being solved for
// end is the value of that component
void travel(struct Point* p, struct Point* v, int xyz, double end_point) {
```

```
  double t=0, initial_point=0, direction_vec=0;

  if (xyz == X_COMPONENT) {
    initial_point = (*p).x;
    direction_vec = (*v).x;
  } else if (xyz == Y_COMPONENT) {
    initial_point = (*p).y;
    direction_vec = (*v).y;
  } else if (xyz == Z_COMPONENT) {
    initial_point = (*p).z;
    direction_vec = (*v).z;
  }

  //the t parameter will give the units needed to move along the vector until the desired end_point is
reached
  t =  (end_point - initial_point) / direction_vec;

  (*p).x = (*p).x + (*v).x * t;
  (*p).y = (*p).y + (*v).y * t;
  (*p).z = (*p).z + (*v).z * t;

  return;
}

struct voxel* track(struct Point* start, struct Point* end, struct Point* volume_end, struct muon*
mu, struct voxel* path, double* M,  double** params, FILE** fps) {

  int xVox, yVox, zVox, curVoxel;
  double vx, vy, vz;
  double tNew, tOld=0, tLast, L, T;
  double xOut = (*start).x, yOut = (*start).y, zOut = (*start).z, depth = 0;

  struct Point *new, *old, *v;
```

```
struct voxel *track;
track = path;

if ((new = (struct Point*) malloc(sizeof(struct Point)))==NULL) return NULL;
if ((old = (struct Point*) malloc(sizeof(struct Point)))==NULL) return NULL;
if ((v   = (struct Point*) malloc(sizeof(struct Point)))==NULL) return NULL;

vec_copy_point(start, new);

vx = (*end).x - (*start).x;
vy = (*end).y - (*start).y;
vz = (*end).z - (*start).z;

while (zOut > (*end).z) {

   xVox = (int) floor(((xOut - *params[PARAM_X_MIN]) /
*params[PARAM_X_VOXEL_SIZE]));
   yVox = (int) floor(((yOut - *params[PARAM_Y_MIN]) /
*params[PARAM_Y_VOXEL_SIZE]));
   zVox = (int) floor(((zOut - *params[PARAM_Z_MIN]) /
*params[PARAM_Z_VOXEL_SIZE]));

   //voxel number determined in z direction first, then y, then x
   curVoxel = xVox * ((int) *params[PARAM_Y_VOXEL_TOTAL]) * ((int)
*params[PARAM_Z_VOXEL_TOTAL]) + yVox * ((int) *params[PARAM_Z_VOXEL_TOTAL])
+ zVox;

   tLast = tOld;
   tOld = 100;

   //calculations for minimum border of voxel in X direction
   xOut = *params[PARAM_X_MIN] + xVox * *params[PARAM_X_VOXEL_SIZE];
   tNew = (xOut-(*start).x)/vx;
```

```
    if ((tNew < tOld) && (tNew > tLast) && (xOut >= *params[PARAM_X_MIN])) tOld=tNew;

    //calculations for maximum border of voxel in X direction
    xOut = *params[PARAM_X_MIN] + (xVox+1) * *params[PARAM_X_VOXEL_SIZE];
    tNew = (xOut-(*start).x)/vx;
    if ((tNew < tOld) && (tNew > tLast) && (xOut <= *params[PARAM_X_MAX])) tOld=tNew;

    //calculations for minimum border of voxel in Y direction
    yOut = *params[PARAM_Y_MIN] + yVox * *params[PARAM_Y_VOXEL_SIZE];
    tNew = (yOut-(*start).y)/vy;
    if ((tNew < tOld) && (tNew > tLast) && (yOut >= *params[PARAM_Y_MIN])) tOld=tNew;

    //calculations for maximum border of voxel in Y direction
    yOut = *params[PARAM_Y_MIN] + (yVox+1)**params[PARAM_Y_VOXEL_SIZE];
    tNew = (yOut-(*start).y)/vy;
    if ((tNew < tOld) && (tNew > tLast) && (yOut <= *params[PARAM_Y_MAX])) tOld=tNew;

    //calculations for minimum border of voxel in Z direction
    zOut = *params[PARAM_Z_MIN] + zVox**params[PARAM_Z_VOXEL_SIZE];
    tNew = (zOut-(*start).z)/vz;
    if ((tNew <tOld) && (tNew > tLast) && (zOut <= *params[PARAM_Z_MAX])) tOld=tNew;

    tNew = tOld + TRACK_PUSH;

    if (*params[PARAM_PRECISE_L] || *params[PARAM_PRECISE_T]) {
      vec_copy_point(new, old);
      (*new).x = (*start).x + vx*tOld;
      (*new).y = (*start).y + vy*tOld;
      (*new).z = (*start).z + vz*tOld;
    }

    if ((zVox < *params[PARAM_Z_VOXEL_TOTAL]) && (zOut < (*start).z) && (zVox >= 0)
&& (curVoxel!= *params[PARAM_PREV_VOXEL])) {
```

```
track=(*track).nextVoxel;
(*track).ID = curVoxel;

if (volume_end!=NULL) (*mu).b = (*mu).b + 1;
else                  (*mu).a = (*mu).a + 1;

//L calculation
if (*params[PARAM_PRECISE_L]) {
  vec_sub(new, old, v);
  L = vec_norm(v);
} else L = *params[PARAM_L];

//T calculation
if (*params[PARAM_PRECISE_T]) {
  if (volume_end==NULL) {
    vec_sub(end, new, v);
    T = vec_norm(v);
  } else {
    vec_sub(end, new, v);
    T = vec_norm(v);
    vec_sub(volume_end, new, v);
    T = T + vec_norm(v);
  }
} else T = *params[PARAM_L] * zVox;

L = L / *params[PARAM_UNITS_LENGTH];
T = T / *params[PARAM_UNITS_LENGTH];

(*track).wt  = L;
(*track).wtX = (pow(L, 2)/2) + L*T;
(*track).wX  = (pow(L, 3)/3) + (pow(L, 2)*T) + (L*pow(T, 2));

if (((*track).nextVoxel = (struct voxel*) malloc(sizeof(struct voxel)))==NULL) return NULL;
```

```
        }

        xOut = (*start).x + vx*tNew;
        yOut = (*start).y + vy*tNew;
        zOut = (*start).z + vz*tNew;

        *params[PARAM_PREV_VOXEL] = curVoxel;

    }

    free((*track).nextVoxel);
    (*track).nextVoxel= NULL;

    free(v);
    free(old);
    free(new);

    return track;
}
```

```
//mtsio.h

//define constants for markers in poca input file
#define TOP_START    'a'
#define TOP_END      'b'
#define BOTTOM_START 'c'
#define BOTTOM_END   'd'
#define EVENTS_START 'e'

int get_geant_input(struct Point**, struct Point**, double**, FILE**);
void write_lambda(double*, double*, struct muon*, double**, FILE*);
void write_optional(struct Point*, struct Point*, struct Point*, double**, FILE**);
```

```
//mtsio.c
#include "mts.h"
#include "mtsio.h"
#include "mtserr.h"
#include "vecfunc.h"

//takes input from a file and parses it into the incoming/outgoing points
//and the scattering angle
int get_geant_input(struct Point** incPoints, struct Point** outPoints, double** params, FILE**
fps) {

  //coordinates from file
  char *token, input[MAX_LINE];
  int i = 0;
  struct Point* dummy;

  while (1) {

    //get next params[PARAM_LINE] from file
    if ((fgets(input, MAX_LINE, fps[FP_IN])==NULL)) return END;
    (*params[PARAM_LINE])++;

    //if the next points aren't from top detector disregard this event

    if (input[0]!=TOP_START) {
      if (input[0]==EVENTS_START) {
        if ((fgets(input, MAX_LINE, fps[FP_IN])==NULL)) return END;
      }
      continue;
    }

    //take care of momentum
    if ((fgets(input, MAX_LINE, fps[FP_IN])==NULL)) return END;
    (*params[PARAM_LINE])++;
```

```
if (!isdigit(input[0]) && input[0]!='-') break;
*params[PARAM_MOMENTUM] = atof(strtok(input, DELIMS));

i = 0;
//read in incoming muon tracks
while (input[0]!=TOP_END) {

    if ((fgets(input, MAX_LINE, fps[FP_IN])==NULL)) return END;
    (*params[PARAM_LINE])++;

    if (!isdigit(input[0]) && input[0]!='-') break;

    dummy = incPoints[i];

    if ((token=strtok( input, DELIMS ))!=NULL) (*dummy).x = atof(token);
    else return formatError("inputFile", input, *params[PARAM_LINE]);
    if ((token=strtok( NULL, DELIMS ))!=NULL) (*dummy).y = atof(token);
    else return formatError("inputFile", input, *params[PARAM_LINE]);
    if ((token=strtok( NULL, DELIMS ))!=NULL) (*dummy).z = atof(token);
    else return formatError("inputFile", input, *params[PARAM_LINE]);

    i++;

}
*params[PARAM_INC_DECT] = i;

//if less than 2 detectors were hit disregard this event
if (i<2) continue;
if (input[0]!=TOP_END) return formatError("inputFile", input, *params[PARAM_LINE]);

if ((fgets(input, MAX_LINE, fps[FP_IN])==NULL)) return END;
(*params[PARAM_LINE])++;
```

```
    //if the next points aren't from bottom detector disregard this event
    if (input[0]!=BOTTOM_START) continue;

    i=0;

    //read in outgoing muon tracks
    while (input[0]!=BOTTOM_END) {

        if ((fgets(input, MAX_LINE, fps[FP_IN])==NULL)) return END;
        (*params[PARAM_LINE])++;

        if (!isdigit(input[0]) && input[0]!='-') break;

        dummy = outPoints[i];

        if ((token=strtok( input, DELIMS ))!=NULL) (*dummy).x = atof(token);
        else return formatError("inputFile", input, *params[PARAM_LINE]);
        if ((token=strtok( NULL, DELIMS ))!=NULL)  (*dummy).y = atof(token);
        else return formatError("inputFile", input, *params[PARAM_LINE]);
        if ((token=strtok( NULL, DELIMS ))!=NULL)  (*dummy).z = atof(token);
        else return formatError("inputFile", input, *params[PARAM_LINE]);

        i++;

    }
    *params[PARAM_OUT_DECT] = i;

    //if less than 2 detectors were hit disregard this event
    if (i<2) continue;
    if (input[0]!=BOTTOM_END) return formatError("inputFile", input, *params[PARAM_LINE]);
    break;


}
```

```
  return CONTINUE;
}


void write_lambda(double* lambda, double* M, struct muon* mu, double** params, FILE* out) {

  int i, x, y, z, startVoxel=0;

  struct voxel* tempVoxel;

  for (i=0; i < *params[PARAM_ALL_VOXELS]; i++) {

    if (M[i]==0) lambda[i]=0;
    if (*params[PARAM_MILLIRADIANS]==1) lambda[i] = lambda[i]*1000000;
    if (*params[PARAM_UNITS_LENGTH]!=10) lambda[i] = lambda[i] * (10 /
*params[PARAM_UNITS_LENGTH]);

    x = (int) floor(i/(*params[PARAM_Y_VOXEL_TOTAL] *
*params[PARAM_Z_VOXEL_TOTAL]));
    y = (int) floor((i - (x * *params[PARAM_Y_VOXEL_TOTAL] *
*params[PARAM_Z_VOXEL_TOTAL]))/ *params[PARAM_Z_VOXEL_TOTAL]);
    z = (int) floor(i - (x * *params[PARAM_Y_VOXEL_TOTAL] *
*params[PARAM_Z_VOXEL_TOTAL]) - (y * *params[PARAM_Z_VOXEL_TOTAL]));

    x = x * *params[PARAM_X_VOXEL_SIZE] + *params[PARAM_X_MIN] +
(*params[PARAM_X_VOXEL_SIZE]/2);
    y = y * *params[PARAM_Y_VOXEL_SIZE] + *params[PARAM_Y_MIN] +
(*params[PARAM_Y_VOXEL_SIZE]/2);
    z = z * *params[PARAM_Z_VOXEL_SIZE] + *params[PARAM_Z_MIN] +
(*params[PARAM_Z_VOXEL_SIZE]/2);

    fprintf(out, "%d %d %d %f %d %f\n", x, y, z, lambda[i], i, M[i]);
```

```
  }

  return;
}

void write_optional(struct Point* in, struct Point* scat, struct Point* out, double** params, FILE**
fps) {

  if (*params[PARAM_OP_OUT]) {

    fprintf(fps[FP_OP_OUT], "Event: %f\n", *params[PARAM_CUR_EVENT]);
    fprintf(fps[FP_OP_OUT], "IN: %f %f %f\n", (*in).x, (*in).y, (*in).z);
    fprintf(fps[FP_OP_OUT], "POCA & DOCA: %f %f %f %f\n", (*scat).x, (*scat).y, (*scat).z,
*params[PARAM_DOCA]);
    fprintf(fps[FP_OP_OUT], "OUT: %f %f %f\n", (*out).x, (*out).y, (*out).z);
    fprintf(fps[FP_OP_OUT], "ANGLE(radians/degrees): %f / %f\n",
*params[PARAM_SCAT_ANG], vec_rad_to_deg(*params[PARAM_SCAT_ANG]));
    fprintf(fps[FP_OP_OUT], "EM (dtX, dtY, dX, dY, mom, L): %f %f %f %f %f %f\n\n",
*params[PARAM_DTX], *params[PARAM_DTY], *params[PARAM_DX],
*params[PARAM_DY], *params[PARAM_MOMENTUM], *params[PARAM_L]);

  }

  if (*params[PARAM_POCA])
    fprintf(fps[FP_POCA_OUT], "%f %f %f %f\n", (*scat).x, (*scat).y, (*scat).z,
vec_rad_to_deg(*params[PARAM_SCAT_ANG]));

  if (*params[PARAM_DIST] && (*params[PARAM_DOCA]!=PARALLEL)) {
    fprintf(fps[FP_DIST_OUT], "%f %f", *params[PARAM_SCAT_ANG],
vec_rad_to_deg(*params[PARAM_SCAT_ANG]));
    fprintf(fps[FP_DIST_OUT], " %f %f %f %f\n", *params[PARAM_DTX],
vec_rad_to_deg(*params[PARAM_DTX]), *params[PARAM_DTY],
vec_rad_to_deg(*params[PARAM_DTY]));
  }
```

```
  return;
}
```

```
//vecfunc.h
//define constants for xyz componets
#define ALL_COMPONENTS -1
#define X_COMPONENT    0
#define Y_COMPONENT    1
#define Z_COMPONENT    2

//define constants for fit function
#define FIT_X    1
#define FIT_Y    0
#define FIT_NONE -1

#define RAD 180/(4.0*atan(1.0))

#define MAX_DETECTORS 5
#define MAX_DIMENSION 2

double vec_dot (struct Point*, struct Point*);
double vec_norm (struct Point*);
void vec_mid (struct Point*, struct Point*, struct Point*);
void vec_sub (struct Point*, struct Point*, struct Point*);
void vec_add (struct Point*, struct Point*, struct Point*);
void vec_mult (double, struct Point*, struct Point*);
void vec_div (double, struct Point*, struct Point*);
void vec_fit (struct Point**, struct Line*, int, int);
double vec_rad_to_deg (double);
double vec_angle (struct Line*, struct Line*, int);
void vec_copy(struct Line*, struct Line*);
```

```
//vecfunc.c
#include "mts.h"
#include "vecfunc.h"

double vec_dot (struct Point *P1, struct Point *P2) { return (((*P1).x * (*P2).x) +  ((*P1).y *
(*P2).y) + ((*P1).z * (*P2).z)); }

double vec_norm (struct Point *v) { return (sqrt(vec_dot(v, v))); }

void vec_mid (struct Point *P1, struct Point *P2, struct Point *scatPt) {

  (*scatPt).x = ((*P1).x+(*P2).x)/2;
  (*scatPt).y = ((*P1).y+(*P2).y)/2,
  (*scatPt).z = ((*P1).z+(*P2).z)/2;

  return;
}

void vec_sub (struct Point *P1, struct Point *P2, struct Point *vec) {

  (*vec).x = ( (*P1).x - (*P2).x );
  (*vec).y = ( (*P1).y - (*P2).y );
  (*vec).z = ( (*P1).z - (*P2).z );

  return;
}

void vec_add (struct Point *P1, struct Point *P2, struct Point *vec) {

  (*vec).x = ( (*P1).x + (*P2).x );
  (*vec).y = ( (*P1).y + (*P2).y );
  (*vec).z = ( (*P1).z + (*P2).z );

  return;
```

```
}

void vec_mult (double sc, struct Point *P1, struct Point *vec) {

  (*vec).x = sc * (*P1).x;
  (*vec).y = sc * (*P1).y;
  (*vec).z = sc * (*P1).z;

  return;
}

void vec_div (double sc, struct Point *P1, struct Point *vec) {

  (*vec).x = (*P1).x / sc;
  (*vec).y = (*P1).y / sc;
  (*vec).z = (*P1).z / sc;

  return;
}

void vec_fit (struct Point** points, struct Line* muonTrack, int detectors, int fitX) {

  double vecXY[MAX_DIMENSION][MAX_DETECTORS];
  double vecZ[MAX_DETECTORS];
  double normal[MAX_DETECTORS][MAX_DIMENSION];
  double lhs[MAX_DIMENSION][MAX_DIMENSION];
  double rhs[MAX_DIMENSION];
  double augmented[MAX_DIMENSION+1][MAX_DIMENSION];
  double lSum,rSum,newXY1,newZ1,newXY2,newZ2,factor,c0,c1;

  int i,j,k;

  struct Point* dummy;
```

```
if (fitX==FIT_NONE) {

  dummy = (*muonTrack).P1;

  (*dummy).x = (*points[0]).x;
  (*dummy).y = (*points[0]).y;
  (*dummy).z = (*points[0]).z;

  dummy = (*muonTrack).P2;

  (*dummy).x = (*points[detectors-1]).x;
  (*dummy).y = (*points[detectors-1]).y;
  (*dummy).z = (*points[detectors-1]).z;

} else {

  if ((fitX == FIT_X) && ((*points[0]).x == (*points[detectors-1]).x)) {
    dummy = (*muonTrack).P1;
    (*dummy).x = (*points[0]).x;
    dummy = (*muonTrack).P2;
    (*dummy).x = (*points[detectors-1]).x;
    return;
  }
  else if ((fitX ==FIT_Y) && ((*points[0]).y == (*points[detectors-1]).y)) {
    dummy = (*muonTrack).P1;
    (*dummy).y = (*points[0]).y;
    dummy = (*muonTrack).P2;
    (*dummy).y = (*points[detectors-1]).y;
    return;
  }

  for (i = 0; i<detectors; i++) {
    vecXY[0][i] = 1;
    if (fitX == FIT_X) vecXY[MAX_DIMENSION-1][i] = (*points[i]).x;
```

```
  else      vecXY[MAX_DIMENSION-1][i] = (*points[i]).y;
  vecZ[i] = (*points[i]).z;
}

for (i = 0; i < MAX_DIMENSION; i++) {
  for (j = 0; j < detectors; j++) {
    normal[j][i] = vecXY[i][j];
  }
}

for (i = 0; i < MAX_DIMENSION; i++) {
  for (j = 0; j < MAX_DIMENSION; j++) {
    lSum = 0;
    rSum = 0;
    for (k = 0; k <detectors; k++) {
      lSum += normal[k][i] * vecXY[j][k];
      rSum += normal[k][i] * vecZ[k];
    }
    lhs[i][j] = lSum;
    rhs[i] = rSum;
  }
}

//create augmented matrix to solve 2x2 system
for (i = 0; i < MAX_DIMENSION; i++) {
  for (j = 0; j < MAX_DIMENSION; j++) {
    augmented[i][j] = lhs[i][j];
  }
  augmented[j][i] = rhs[i];
}

//create augmented matrix to solve 2x2 system
augmented[0][0] = lhs[0][0];
augmented[0][1] = lhs[0][1];
```

```
    augmented[1][0] = lhs[1][0];
    augmented[1][1] = lhs[1][1];
    augmented[2][0] = rhs[0];
    augmented[2][1] = rhs[1];

    factor = augmented[0][1]/augmented[0][0];
    for (i = 0; i < 3; i++) {
      augmented[i][1] = augmented[i][1] - (factor*augmented[i][0]);
    }

    c1 = augmented[2][1] / augmented[1][1];
    c0 = (augmented[2][0] - augmented[1][0]*c1) / augmented[0][0];

    newZ1 = vecZ[0];
    newXY1 = (newZ1-c0)/c1;

    newZ2 = vecZ[detectors-1];
    newXY2 = (newZ2-c0)/c1;

    dummy = (*muonTrack).P1;

    if (fitX==FIT_X) (*dummy).x = newXY1;
    else      (*dummy).y = newXY1;
    (*dummy).z = newZ1;

    dummy = (*muonTrack).P2;

    if (fitX==FIT_X) (*dummy).x = newXY2;
    else      (*dummy).y = newXY2;
    (*dummy).z = newZ2;
  }

  return;
}
```

```
double vec_rad_to_deg (double angle) { return (RAD * angle); }

//vec_angle uses a.b=|a||b|acos(theta) to compute the angle between two vectors
double vec_angle (struct Line* L1, struct Line* L2, int component) {

  double scatAng, distance, dotUV, normU, normV;

  struct Point *u,*v;

  if ((u = (struct Point*) malloc(sizeof(struct Point)))==NULL) return memError();
  if ((v = (struct Point*) malloc(sizeof(struct Point)))==NULL) return memError();

  vec_sub((*L1).P2, (*L1).P1, u);
  vec_sub((*L2).P2, (*L2).P1, v);

  if (component!=ALL_COMPONENTS) {
    if (component==X_COMPONENT) (*u).x = (*v).x = 0;
    if (component==Y_COMPONENT) (*u).y = (*v).y = 0;
  }

  normU = vec_norm(u);
  normV = vec_norm(v);

  dotUV = vec_dot(u, v);

  if (((dotUV) / ((normU) * (normV))) >= 1) scatAng = 0;
  else scatAng = acos(dotUV / (normU * normV));

  if (dotUV<0) {
    if (scatAng > 0) scatAng+=M_PI;
    else          scatAng-=M_PI;
  }
```

```
  free(v);
  free(u);

  return scatAng;
}

//vecCopy copies the contents of L1 to L2
void vec_copy(struct Line* L1, struct Line* L2) {

  struct Point *P1, *P2;

  P1 = (*L1).P1;
  P2 = (*L2).P1;

  (*P2).x = (*P1).x;
  (*P2).y = (*P1).y;
  (*P2).z = (*P1).z;

  P1 = (*L1).P2;
  P2 = (*L2).P2;

  (*P2).x = (*P1).x;
  (*P2).y = (*P1).y;
  (*P2).z = (*P1).z;

  return;

}

void vec_copy_point(struct Point* P1, struct Point* P2) {

  (*P2).x = (*P1).x;
  (*P2).y = (*P1).y;
  (*P2).z = (*P1).z;
```

```
  return;
}
```

```
//mtserr.h

//define constants for error return values
#define ERR_CODE_COMMAND_LINE       1
#define ERR_CODE_EMPTY_FILE         2
#define ERR_CODE_INVALID_COMMAND    3
#define ERR_CODE_INVALID_FILE       4
#define ERR_CODE_INVALID_INPUT      5
#define ERR_CODE_MEM                6
#define ERR_CODE_NO_FILE            7
#define ERR_CODE_OPTIONAL_REQUIRED  8
#define ERR_CODE_UNIX               9
#define ERR_CODE_UNUSED_FILE        10

char commandError(char, char);
int emptyError(char*);
int fileError(char*);
int formatError(char*, char*, int);
int memError();
int optionalError(char*, char);
int unixError(char*);
int unusedError(char*);
```

```
//mtserr.c
#include "mts.h"
#include "mtsio.h"

//constants defining the error messages to display for the command line
#define ERR_MSG_TITLE    "Muon Tomography Suite Command Line Options"
#define ERR_MSG_OPTION   "Illegal Command Line Option"
#define ERR_MSG_COMBO    "The following commands may not be selected together"

#define ERR_MSG_COVERAGE1 "run coverage analysis after reconstruction"
#define ERR_MSG_COVERAGE2 "(input file must be provided unless poca/em is being run or
stdin option is chosen)"
#define ERR_MSG_DETAILS1  "provides detailed information on inner processes"
#define ERR_MSG_DETAILS2  0
#define ERR_MSG_EM1       "run maximum likelihood algorithm"
#define ERR_MSG_EM2       "(input file must be provided unless stdin option is selected)"
#define ERR_MSG_HELP1     "list command line options"
#define ERR_MSG_HELP2     0
#define ERR_MSG_NO_FIT1   "run reconstrunction without line fitting the data"
#define ERR_MSG_NO_FIT2   0
#define ERR_MSG_NORM1     "run reconstrunction with normalized vectors"
#define ERR_MSG_NORM2     0
#define ERR_MSG_POCA1     "run poca algorithm"
#define ERR_MSG_POCA2     "(input file must be provided unless stdin option is selected)"
#define ERR_MSG_ROOT1     "run root analysis after reconstruction (cannot be selected with
stdout option)"
#define ERR_MSG_ROOT2     0
#define ERR_MSG_STDIN1    "accept input from stdin instead of file"
#define ERR_MSG_STDIN2    "(all input must come from stdin; provided input files are ignored)"
#define ERR_MSG_STDOUT1   "print output to stdout instead of file"
#define ERR_MSG_STDOUT2   0
#define ERR_MSG_VALIDATE1 "run validation analysis after reconstruction"
#define ERR_MSG_VALIDATE2 "(input file must be provided unless poca/em is being run or stdin
option is chosen)"
```

```
#define ERR_MSG_X1       "extended details, prints algorithmic computations step by step"
#define ERR_MSG_X2       0

//constants defining the error messages for internal MTS runtime errors
#define ERR_MSG_EMPTY    "is empty!"
#define ERR_MSG_FILE     "is not a valid file!"
#define ERR_MSG_FORMAT1  "is incorrectly formatted!"
#define ERR_MSG_FORMAT2  "Invalid Line"
#define ERR_MSG_MEM      "insufficient memory available"
#define ERR_MSG_OPTIONAL "An input file must be provided for the following options"
#define ERR_MSG_UNIX     "unix system command failed:"
#define ERR_MSG_UNUSED   "unused: Input from stdin"

char commandError(char option, char option2) {

  if (option!=option2) printf("\n%s: -%c and -%c\n", ERR_MSG_COMBO, option, option2);
  else if (option!=SHORT_HELP) printf("\n%s: |%c|\n", ERR_MSG_OPTION, option);

  printf("\n%s:\n\n", ERR_MSG_TITLE);
  printf("\t-%c or --%s: \n\n\t\t%s \n\n",        SHORT_DETAILS,  LONG_DETAILS,
ERR_MSG_DETAILS1,  ERR_MSG_DETAILS2);
  printf("\t-%c or --%s: \n\n\t\t%s \n\t\t%s \n\n", SHORT_EM,      LONG_EM,
ERR_MSG_EM1,      ERR_MSG_EM2);
  printf("\t-%c or --%s: \n\n\t\t%s \n\n",        SHORT_HELP,    LONG_HELP,
ERR_MSG_HELP1,    ERR_MSG_HELP2);

  return option;
}

int emptyError(char* file) {
  fprintf(ERR_OUT, "\n\n\t%s %s\n\n", file, ERR_MSG_EMPTY);
  return ERR_CODE_EMPTY_FILE;
}
```

```
int fileError(char* file) {
  fprintf(ERR_OUT, "\n\n\t%s %s!\n\n", file, ERR_MSG_FILE);
  return ERR_CODE_INVALID_FILE;
}

int formatError(char* file, char* input, int line) {
  fprintf(ERR_OUT, "\n\n\t%s %s\n\t\t%s %d: %s\n\n", file, ERR_MSG_FORMAT1,
ERR_MSG_FORMAT2, line, input);
  return ERR_CODE_INVALID_INPUT;
}

int memError() {
  fprintf(ERR_OUT, "\n\n\t%s\n\n", ERR_MSG_MEM);
  return ERR_CODE_MEM;
}

int optionalError(char* longOp, char shortOp) {
  fprintf(ERR_OUT, "\n\n\t%s: -%c and --%s\n\n", ERR_MSG_OPTIONAL, shortOp, longOp);
  return ERR_CODE_OPTIONAL_REQUIRED;
}

int unixError(char* command) {
  fprintf(ERR_OUT, "\n\n\t%s %s\n\n", ERR_MSG_UNIX, command);
  return ERR_CODE_UNIX;
}

int unusedError(char* file) {
  fprintf(ERR_OUT, "\n\t%s %s\n\n", file, ERR_MSG_UNUSED);
  return ERR_CODE_UNUSED_FILE;
}
```

```
//em.h
//define standard extensions
#define EXT_EM        "em"
#define EXT_LT        "lt"
#define EXT_OUT       "out"
#define EXT_MED       "med"
#define EXT_AVG       "avg"
#define EXT_MUON_DATA  "mu"
#define EXT_VOXEL_DATA "vox"
#define EXT_V         "vvals"
#define EXT_C         "cvals"
#define EXT_CONVERGE   "conv"
#define EXT_LAMBDAS    "lam"
#define EXT_SAMP_VOX   "smp"

//define standard constants
#define MAX_ITER      2 //for debugging purposes, not computational

#define RAD_CONVERT   1000  //standard (=1) is radians
#define LENGTH_CONVERT 10 //standard (=1) is millimeters

#define X 0
#define Y 1
#define Z 2

//#define PRINT_ITERATION i>0
//#define PRINT_ITERATION ((i+1)%5)==0
#define PRINT_ITERATION ((i+1)%10)==0

int em(double*, double*, struct muon*, double**, FILE**);
void compute_c(struct muon*, double*, double*, int, double**, double**, double**);
void compute_v(struct muon*, double*, double*, double*, int, double**);
double calc_voxel_weight(struct muon*, unsigned int, double, double**);
```

```
//em.c
#include "mts.h"
#include "em.h"


int em (double* lambdaMed, double* lambda, struct muon* head, double** params, FILE** fps) {

  fprintf(stderr, "\n\n%s EM %s\n\n", BANNER, BANNER);

  int i, j, k, bin, allVoxels, iterations, done=0;
  double lambdaTemp, *M, *C, **Cbin, **Cn, bCount;
  time_t startLocal, endLocal;
  struct muon *tempMuon;

  allVoxels = *params[PARAM_ALL_VOXELS];

  if ((M=(double*) malloc(allVoxels*sizeof(double)))==NULL) return memError();

  if (*params[PARAM_EM_AVERAGE]) {
    if (!*params[PARAM_EM_ONLINE] || ((*params[PARAM_CUR_EVENT] /
*params[PARAM_EM_ONLINE])==1)) {
      if ((lambda=(double*) malloc(allVoxels*sizeof(double)))==NULL) return memError();
    }
    if ((C=(double*) malloc(allVoxels*sizeof(double)))==NULL) return memError();
  }

  if (*params[PARAM_EM_MEDIAN]) {

    if (!*params[PARAM_EM_ONLINE] || ((*params[PARAM_CUR_EVENT] /
*params[PARAM_EM_ONLINE])==1)) {
      if ((lambdaMed=(double*) malloc(allVoxels*sizeof(double)))==NULL) return memError();
    }
    if ((Cbin=(double**) malloc(allVoxels*sizeof(double)))==NULL) return memError();
    if ((Cn=(double**) malloc(allVoxels*sizeof(double)))==NULL) return memError();
```

```
    for (i=0; i<*params[PARAM_ALL_VOXELS]; i++) {
      if ((Cbin[i]=(double*) malloc(*params[PARAM_EM_BINS]*sizeof(double)))==NULL)
return memError();
      if ((Cn[i]=(double*) malloc(*params[PARAM_EM_BINS]*sizeof(double)))==NULL) return
memError();
    }
  }

  for (i=0;i<*params[PARAM_ALL_VOXELS];i++) {
    M[i]=0;
    if (!*params[PARAM_EM_ONLINE] || ((*params[PARAM_CUR_EVENT] /
*params[PARAM_EM_ONLINE])==1)) {
      if (*params[PARAM_EM_AVERAGE]) lambda[i]=*params[PARAM_INIT_LAMBDA];
    }
    if (*params[PARAM_EM_MEDIAN]) {
      if (!*params[PARAM_EM_ONLINE] || ((*params[PARAM_CUR_EVENT] /
*params[PARAM_EM_ONLINE])==1)) {
        lambdaMed[i]=*params[PARAM_INIT_LAMBDA];
      }
      for (j=0; j<*params[PARAM_EM_BINS]; j++) {
        Cbin[i][j]=0;
        Cn[i][j]=0;
      }
    }
  }

  char fnVoxel[100];

  fprintf(stderr, "\nEntering EM Main Loop...\n");

  time(&startLocal);
  iterations = *params[PARAM_ITERATIONS];
  for (i=0; i<iterations; i++) {
```

```
    fprintf(stderr, "\n\tIteration %d of %d ", (i+1), iterations);


    if (*params[PARAM_EM_AVERAGE]) memset(C, '\0', sizeof(double) *
*params[PARAM_ALL_VOXELS]);

    tempMuon = head;
    while ((tempMuon=(*tempMuon).nextMuon)!=NULL) {

      compute_v(tempMuon, M, lambda, lambdaMed, i, params);
      compute_c(tempMuon, C, lambdaMed, i, Cbin, Cn, params);
    }

    for (j=0; j<allVoxels; j++) {

      if (M[j]!=0) {

        if (*params[PARAM_EM_AVERAGE]) {
          lambdaTemp = lambda[j];
          lambda[j] = lambda[j] + pow(lambda[j], 2) * (1/M[j]) * C[j];
          if (lambda[j]<=0) lambda[j]=*params[PARAM_INIT_LAMBDA];
        }

        if (*params[PARAM_EM_MEDIAN]) {

          bin = 0;
          for (k=0, bCount=0; k<*params[PARAM_EM_BINS]; k++) {
            bCount = bCount + Cn[j][k];
            if (bCount > (M[j]/2)) {
              bin = k;
              break;
            }
          }
```

```
          lambdaMed[j] = 0.5 * (Cbin[j][bin]/Cn[j][bin]);
          if (lambdaMed[j]<=0) lambdaMed[j]=*params[PARAM_INIT_LAMBDA];

          for (k=0; k<*params[PARAM_EM_BINS]; k++) {
            Cbin[j][k]=0;
            Cn[j][k]=0;
          }
        }
      }
    }

  }

  time(&endLocal);
  fprintf(stderr, "\n\nEM Program ran for %f seconds\n\n", difftime(endLocal, startLocal));

  if (!*params[PARAM_EM_ONLINE] || (fmod(*params[PARAM_CUR_EVENT],
*params[PARAM_EM_ONLINE])!=0)) {
    if (*params[PARAM_EM_AVERAGE]) write_lambda(lambda, M, NULL, params,
fps[FP_OUT_AVG]);
    if (*params[PARAM_EM_MEDIAN]) write_lambda(lambdaMed, M, NULL, params,
fps[FP_OUT_MED]);
  }

  free(M);
  if (*params[PARAM_EM_AVERAGE]) free(C);
  if (!*params[PARAM_EM_ONLINE] || (fmod(*params[PARAM_CUR_EVENT],
*params[PARAM_EM_ONLINE])!=0)) {
    if (*params[PARAM_EM_AVERAGE]) free(lambda);
    if (*params[PARAM_EM_MEDIAN]) free(lambdaMed);
  }

  if (*params[PARAM_EM_MEDIAN]) {
```

```
    for (i=0;i<*params[PARAM_ALL_VOXELS];i++) {
      free(Cbin[i]);
      free(Cn[i]);
    }
  }
  free(Cbin);
  free(Cn);

  return;
}

void compute_v (struct muon* mu, double* M, double* lambda, double* lambdaMed, int iteration,
double** params) {

  double det, sigma0, sigma1, sigma2, I[4], c=0, weight=1, noInv=0, ID0, ID1, ID2, ID3;
  struct voxel *track;

  if (*params[PARAM_EM_AVERAGE]) {
    (*mu).sigma[0]=0;
    (*mu).sigma[1]=0;
    (*mu).sigma[2]=0;
  }

  if (*params[PARAM_EM_MEDIAN]) {
    (*mu).sigmaMed[0]=0;
    (*mu).sigmaMed[1]=0;
    (*mu).sigmaMed[2]=0;
  }

  track = (*mu).muonTrack;
  do {

    c = c + 1;
    if (*params[PARAM_EM_WEIGHTED]) weight = calc_voxel_weight(mu, (*track).ID, c,
```

```
params);

      if (iteration==0) M[(*track).ID] = M[(*track).ID] + 1;
      if (*params[PARAM_EM_AVERAGE]) {
        (*mu).sigma[0] = (*mu).sigma[0] + weight * (*track).wt  * lambda[(*track).ID];
        (*mu).sigma[1] = (*mu).sigma[1] + weight * (*track).wtX * lambda[(*track).ID];
        (*mu).sigma[2] = (*mu).sigma[2] + weight * (*track).wX  * lambda[(*track).ID];
      }

      if (*params[PARAM_EM_MEDIAN]) {
        (*mu).sigmaMed[0] = (*mu).sigmaMed[0] + weight * (*track).wt  * lambdaMed[(*track).ID];
        (*mu).sigmaMed[1] = (*mu).sigmaMed[1] + weight * (*track).wtX *
lambdaMed[(*track).ID];
        (*mu).sigmaMed[2] = (*mu).sigmaMed[2] + weight * (*track).wX  *
lambdaMed[(*track).ID];
      }

    } while ((track=(*track).nextVoxel)!=NULL);


    if (*params[PARAM_EM_AVERAGE]) {
      (*mu).sigma[0] = (*mu).sigma[0] * (*mu).pr2;
      (*mu).sigma[1] = (*mu).sigma[1] * (*mu).pr2;
      (*mu).sigma[2] = (*mu).sigma[2] * (*mu).pr2;


      if ((det = ((*mu).sigma[0] * (*mu).sigma[2]) - ((*mu).sigma[1] * (*mu).sigma[1]))==0) {
        fprintf(stderr, "\nError Singular Matrix: Event %d\n\n", (*mu).event);
        det = 0;
      }

      sigma0 = (*mu).sigma[0];
      sigma1 = (*mu).sigma[1];
      sigma2 = (*mu).sigma[2];
```

```
    (*mu).sigma[0] = (1/det) *  sigma2;
    (*mu).sigma[1] = (1/det) *  sigma1 * -1;
    (*mu).sigma[2] = (1/det) *  sigma0;

  }

  if (*params[PARAM_EM_MEDIAN]) {

    (*mu).sigmaMed[0] = (*mu).sigmaMed[0] * (*mu).pr2;
    (*mu).sigmaMed[1] = (*mu).sigmaMed[1] * (*mu).pr2;
    (*mu).sigmaMed[2] = (*mu).sigmaMed[2] * (*mu).pr2;

    if ((det = ((*mu).sigmaMed[0] * (*mu).sigmaMed[2]) - ((*mu).sigmaMed[1] *
(*mu).sigmaMed[1]))==0) {
      fprintf(stderr, "\nError Singular Matrix: Event %d\n\n", (*mu).event);
      det = 0;
    }

    sigma0 = (*mu).sigmaMed[0];
    sigma1 = (*mu).sigmaMed[1];
    sigma2 = (*mu).sigmaMed[2];

    (*mu).sigmaMed[0] = (1/det) *  sigma2;
    (*mu).sigmaMed[1] = (1/det) *  sigma1 * -1;
    (*mu).sigmaMed[2] = (1/det) *  sigma0;
  }

  return;
}

double calc_voxel_weight(struct muon* mu, unsigned int ID, double c, double** params) {

  double a, b, d;
```

```
  if ((*mu).b==0) return 1;

  a = (double) (*mu).a; //voxels after scatter voxel
  b = (double) (*mu).b; //voxels before scatter voxel
  d = b + 1;           //voxel of scattering

  if (*params[PARAM_EM_WEIGHTED]==1) {
    if (c!=d) return 0;
    else      return 1;
  }

  if (*params[PARAM_EM_WEIGHTED]==2) {

    if (a==0 || b==0) return 1;
    if (c<d)       return ((b - (d - c)) / b);
    else if (c==d) return 1;
    else           return ((a - (c - d)) / a);

  }

  return 1;
}

void compute_c (struct muon* mu, double* C, double* lambdaMed, int iteration, double** Cbin,
double** Cn, double** params) {

  int ID, ibin, i;
  double dtX, dtY, dX, dY, pr2, fbin, bin_size = 0, neg_bin_size = 2, big_bin_size = 10000;
  double wt, wtX, wX, v11, v12, v22, v11m, v12m, v22m, a, b, c, ax, bx, cx, ay, by, cy, oldC, newC;
  struct voxel *track, *trackTemp, *trackMed;

  dtX = (*mu).dtX;
  dtY = (*mu).dtY;
```

```
dX  = (*mu).dX;
dY  = (*mu).dY;
pr2 = (*mu).pr2;

v11 = (*mu).sigma[0];
v12 = (*mu).sigma[1];
v22 = (*mu).sigma[2];

v11m = (*mu).sigmaMed[0];
v12m = (*mu).sigmaMed[1];
v22m = (*mu).sigmaMed[2];

track = (*mu).muonTrack;
do {

  ID = (*track).ID;

  wt  = (*track).wt;
  wtX = (*track).wtX;
  wX  = (*track).wX;

  if (*params[PARAM_EM_AVERAGE]) {

    ax = dtX * v11 + dX * v12;
    bx = dtX * v12 + dX * v22;
    cx = ax * bx;

    ay = dtY * v11 + dY * v12;
    by = dtY * v12 + dY * v22;
    cy = ay * by;

    a = (pow(ax, 2) + pow(ay, 2)) / *params[PARAM_EM];
    b = (pow(bx, 2) + pow(by, 2)) / *params[PARAM_EM];
    c = (cx + cy) / *params[PARAM_EM];
```

```
    oldC = C[ID];
    newC = ((pr2 * (a - v11) * wt) + (pr2 * 2 * (c - v12) * wtX) + (pr2 * (b - v22) * wX));
    C[ID] = C[ID] +  newC;

  }

  if (*params[PARAM_EM_MEDIAN]) {

    ax = dtX * v11m + dX * v12m;
    bx = dtX * v12m + dX * v22m;
    cx = ax * bx;

    ay = dtY * v11m + dY * v12m;
    by = dtY * v12m + dY * v22m;
    cy = ay * by;

    a = (pow(ax, 2) + pow(ay, 2)) / *params[PARAM_EM];
    b = (pow(bx, 2) + pow(by, 2)) / *params[PARAM_EM];
    c = (cx + cy) / *params[PARAM_EM];

    newC = ((pr2 * (a - v11m) * wt) + (pr2 * 2 * (c - v12m) * wtX) + (pr2 * (b - v22m) * wX));

    bin_size = *params[PARAM_EM_BIN_SIZE];
    fbin = (newC / bin_size) + (*params[PARAM_EM_BINS]/2) + 1;

    if (fbin < 0) fbin = 0;
    else if (fbin >= *params[PARAM_EM_BINS]) fbin = *params[PARAM_EM_BINS] - 1;

    ibin = ((int) floor(fbin));

    Cbin[ID][ibin] = Cbin[ID][ibin] + (2*lambdaMed[ID] +  pow(lambdaMed[ID], 2) * newC);
    Cn[ID][ibin]   = Cn[ID][ibin] + 1;
```

```
    }

  } while ((track=(*track).nextVoxel)!=NULL);

  return;
}
```