# Efficiency Analysis of Track Reconstruction From Monte Carlo Simulations of Particle Collisions

A Thesis Submitted in Satisfaction of the Requirements for the Degree of Bachelor of Science in Physics at the University of California, Santa Cruz.

By Jeff Schulte

May 1, 2009

Dr. Bruce Schumm

Thesis and Technical Advisor

David P. Belanger
Chair, Department of Physics

## Contents

# Introduction and overview

Much of the story of physics throughout the past hundred years has been about extremes. Physicists have been exploring the laws of nature at smaller and smaller levels, at higher and higher energies, and have been investigating further and further out into the universe. The LHC, just recently completed, has been the latest big leap in physicist's ability to study some of these extremes. Colliding protons together at higher energies than has ever been accessible before and allowing physicists to study the particles that are emitted from these explosions, the LHC will shed a great deal of light on the nature of matter and the origins of the universe. The proposed International Linear Collider (ILC), the focus of this thesis, will be a compliment to this machine. It will be a more high-precision machine and will in the end be able to operate at levels of energy in the one trillion electron-volt (TeV) range.

The ILC will consist of two linear accelerators facing each other, stretching over a total of 30 kilometers. Instead of two proton beams, the ILC will accelerate an electron beam and a positron beam towards each other, and the particles in the two beams will collide, 10,000 times every second, at the center of a large detector. A central component of this is a silicon detector which allows scientists to record the paths of particles, in a sense. Although the detector cannot give direct information regarding the entire path of any one particle, it can record a "hit" where a particle has produced ionization. There's also no information the detector itself can give regarding which particle created any one hit.

The data that the analysis computer software will work on than will essentially be a group of hits spread out over a map of this detector. It is up to the packages of code to reconstruct "tracks" out of this group – paths of particles through the detector, characterized by a small number of varying parameters. These specific parameters will be discussed in what follows.

The general goal of this project is to enhance a C++ package of code that will analyze the efficiency of the track reconstruction process, and to use this to study the capabilities of the proposed LC detector's tracking system.

## Track reconstruction and the creation of the flat file

The package that reconstructs the tracks is very extensive, is written in JAVA, and represents the work of many scientists. In it is a high level pattern recognition algorithm written by Richard Partridge at SLAC. This package is loaded into an application, JAS3, which is designed especially for particle physics applications. Within this application, driver files that reside in the larger package can be run. These driver files make calls to other utility routines in the package, which then call upon other routines, etc. It is in the driver routine that the final "process" method will reside. The idea is that this "process" method runs over a raw data file that is also loaded into JAS3. This raw data file holds within it "the mess" (of detector hits), if you will.

Also in this driver file will be a write-out statement which will organize the "processed" data and then print it out onto what becomes the end result of the JAVA code for my project, the flat file. This flat file is specifically designed for the C++ code to run over and make its efficiency analysis. The flat file is a very large file of ASCII values, sorted by column and row. Each row is comprised of a series of printed values that detail the attributes of one individual particle or reconstructed particle track.

A significant point to make with respect to this flat file is that it holds information about the tracks that have been reconstructed by the JAVA code and as well information about the "actual" particles. This needs to be explained. When the ILC is up and running, there will only be, as stated above, information that details where hits have been created within the detector. In the actual experiment, there cannot be any "God-given" information about the real particle paths. While scientists prepare for real reconstruction data, and attempt to maximize the performance of their software, they create instead Monte Carlo simulations of data that the detector will eventually build. In so doing, however, they are able to create and record information that the real detector would never yield. In particular they are able to record the “real particle track” of a simulated particle’s path through the detector. In other words, the MC simulation creates a “real” particle track, and then creates the hits that such a track would leave behind, and then is able to give both of these types of information to the reconstruction software. When, in our driver in the JAVA package, we write information to a flat file that will later be analyzed, we write out information about each track that has been reconstructed and as well information about each real particle that has been simulated. The C++ analysis package can at that point run over the flat file, and thanks to this “God-given” information, it can compare the reconstructed tracks to the actual particles. This is the basis on which it is able to make judgments about the efficiency of the reconstruction package, which is indeed the reason the C++ package exists.

## C++ Analysis Package and its use of the flat file

The C++ package that analyzes the reconstruction is built around a series of parameters, taken from the flat file, that detail different properties of the particles paths and reconstructed tracks taken from the raw data. These properties are then used in a series of cuts that sort out the paths and tracks into different categories, and different histograms of these properties use each of these categories. The histograms that are printed out by the C++ code depict the “real results” of the effectiveness of the detector. The following details how the particles and tracks are sorted into categories, and then how the parameters are used in the sorting.

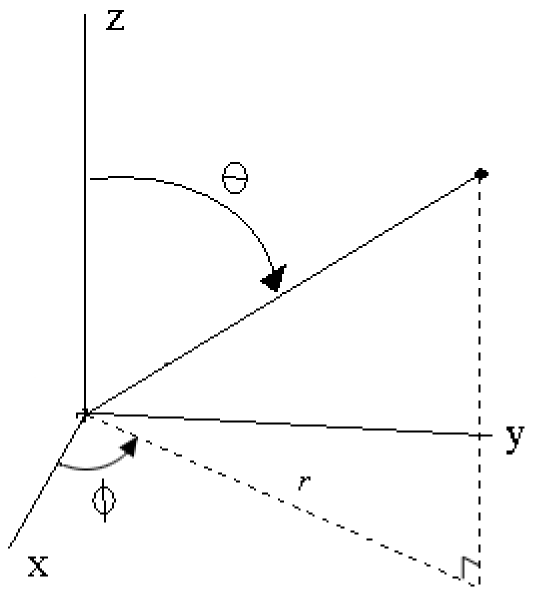
The flat file is a very large file of ASCII values, sorted by column and row. Each row is comprised of a series of printed values that detail the attributes of one individual particle or track. One of the things that the C++ code does is to run through each particle row and pick out which among them are labeled as “findable”.  The idea is that these particles, picked out from the many others, are the particles that we expect a good working reconstruction package to find tracks for.  In other words, for a perfectly efficient reconstruction, every one of these findable particles would have an associated track, also recorded in the flat file, which would have similar characteristics to the particle in question.

The analysis package will also sort the flat file rows representing reconstructed tracks into “acceptable” tracks and “non-acceptable” tracks, once again using parameter cuts that the programmer can specify.  The idea here is that the reconstruction process loses efficiency when certain parts of the detector are being run through by the track.  For example, we may specify that any track which is labeled as “acceptable” must have a |Cos $\theta$| which is below 0.7.  This angle ($\theta$), which we will discuss later, will in essence tell us how close to the original colliding particle beam the track is when it runs through the detector.  A |Cos $\theta$| of 1.0 would represent a track that is running directly along the original particle beam.  We can imagine (and it is certainly true) that when a particle that we’re trying to track is so close to the beam line, there is a lot of confusing back round noise, and it will be harder to reconstruct there.  To avoid this, we simply say that we don’t even want to look at particles which come off at that angle.  The programmer sets the parameters so that both when deciding whether particles are “findable” and when tracks are “acceptable”, we refuse to accept those particles and tracks that run off at an angle so close to the original particle beam.  Thus, we use parameters to decide in essence which particles and tracks to even look at, which are “findable” and “acceptable”, respectively.

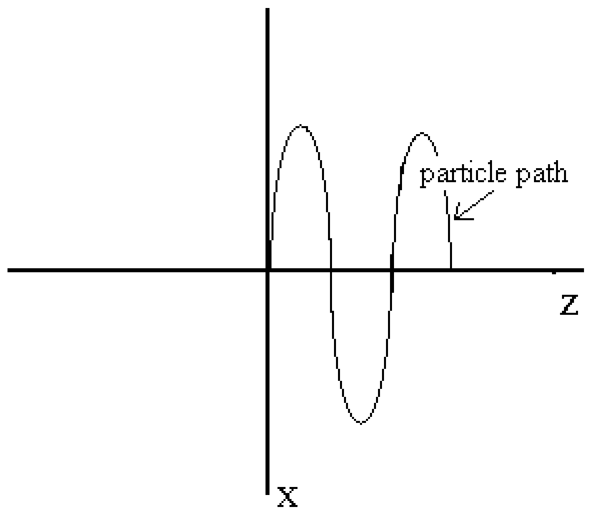
## Definition of parameters

In order to explain in detail what parameters are used in deciding which particles are “findable”, which are “acceptable”, and how the purity and efficiency histograms are constructed, we list and explain below the primary parameters that are printed out into the flat file, and then go on to explain how these parameters are used in the grouping cuts.

We start by setting up a coordinate system which explains many of the angles below. Figure 1, shown below, shows a three dimensional coordinate system. The z-axis is the axis that depicts the original colliding particle beam. Both colliding beams will run along this axis and the collision will occur at the origin. Using spherical coordinates, than, our $\phi$ represents the angle relative to the x axis in the x-y plane, while our theta angle is that which measures the separation between the z-axis and the particle path in question. When Cos $\theta$ of the particle is equal to zero, then $\phi$ will run around in the plane that is perpendicular to the colliding beams. Also, as we stated earlier, if a particle has a Cos $\theta$ of exactly 1.0, the particle runs parallel to the original colliding beam. If the Cos $\theta$ is zero, the particle runs in the plane that is perpendicular to the beam. We can see that if we want to look at particles which move away from the beam, we must make a cut on the maximum Cos $\theta$ that we're willing to look at.



The next diagram shows the same coordinates but from the point of view of an observer who is looking down the y axis.



The particles will trace out a “helical” trajectory in the detector. This means that its velocity along the z axis will be constant, while the x-y projection of its trajectory will be a circle. This circle is created by a constant magnetic field that is sustained throughout the detector and that points along the z axis. In the x-y plane, when particles move away from the origin, they move perpindicularly to this magnetic field and are thus accelerated so that they curve with a constant radius of curvature. The equation that dictates this curvature in $m^{-1}$ is:

$$\omega = \frac{1}{R} = \frac{300B}{pc}$$

Where “pc”, the momentum of the particle, is measured in MeV and “B”, the magnetic field, in Tesla. This then dictates the radius of curvature of the particle path after it has been projected onto the x-y plane. The z-component of the particle path undergoes no acceleration since the magnetic field is set along the z-axis.

One more thing to define in this coordinate system and layout is the angle lambda, or $\lambda$. The dip angle lamda measures the angle between the particle trajectory projected in the x-y plane and the three dimensional trajectory, and is related to $\theta$ via $x = \frac{\pi}{2} - \theta$. The tangent of lambda , which is used in our definition of the pathlength of the particle, can be seen to be equal to the z-momentum component divided by the x-y plane momentum. This is discussed further in the pathlength definition below.

## List of Parameters in the Flat File

EVENT_I

This keeps track of which event number we're dealing with (which collision). The flat file prints out, in linear order, information from every collision in the raw data file. For each of these there are a number of particles and tracks.

PCOSTHETA_I

TCOSTHETA_I

Cosine of theta has been discribed above. It is the angle that denotes how close to the original particle beam the particle or track is. Almost all of these paremters are broken up into P and T parameters. These, as one might guess, refer to whether the object in question is a particle or a reconstructed particle track. Both are described with the same set of parameters, but for the tracks, the particle-specific permaters simply aren't filled, and for the particles, the track-specific peramters aren't filled. A track, then, will list a 0.0 for PCOSTHETA_I in its column in the flat file. It will list its actaul cosine of theta in the TCOSTHETA_I column of the flat file.

PALPHA_I

TALPHA_I

Alpha is an angle that is related to the "jet's" cosine of theta. The original collision of the two particles beams will send off two "jets", which originally begin as two quarks, in opposite directions in the detector. These jets will quickly split up and become collimated showers of particles. There is, however, computed in the JAVA code, an average 3-d angle that these showers of particles are coming out at within the detector. The alpha angle, for every individual particle and track, describes the angle that the particle or track comes off at with respect to this jet theta angle. In other words, it measures how far away from the general direction of the shower the particle originally comes out at - its deviation from the primary jet direction.

PPHI_I

TPHI_I

As mentioned above, Phi is the angle that revoles in the plane that is perpindicualr to the particle beams. It passes through the origin, or the original point of collision. Each particle and track is further specified by the angle Phi at which it emerges.

POMEGA_I

TOMEGA_I

Omega is a measure of the amount of curvature of each particle and track. It is defined above in the explanation of the helical trajectories and the magnetic field.

PPATHLENGTH_I

TPATHLENGTH_I

Pathlength is a measure of the how far the particle or track travels in its path through the detector. Specifically it measures how much distance the particle path covers after it has been projected onto the x-y plane. As stated above, the tangent of lambda is defined as

$$tan\lambda = \frac{p_z}{p_T},$$

where the numerator is the z-component of momentum and the denominator is the transverse momentum. The pathlength $\Delta s$ corresponding to an axial displacement of $\Delta z$ is given by

$$\Delta s = \frac{\Delta z}{tan\lambda},$$

which we can see represents the length of the incremental change in the length of the trajectory.

PPT_I

TPT_I

The $p_T$ is a measurement of the momentum of the particle in the x-y plane. These $p_T$ readings are computed in the JAVA code, using both the radius of curvature calculations and the set magnetic field strength.

PURITY_I

The purity rating is discussed below, in the section regarding fake tracks (fake tracks themselves are defined below). It represents, for every track, the percentage of hits on that track which actually came from the track's associated particle.

INDEX_I

This is an integer number assigned to every particle and track in the event, assigned so that they can be refered to individually in the code. Every particle is assigned a negative integer and every track is assigned a positive one.

ASSINDEX_I

This is very similar to INDEX_I. Hoever, it labels not the particle or tracks that it refers to, but rather the INDEX_I label of the particle or track that is associated with the one in question. For example, if I were to look at the ASSINDEX_I of a track, and that track was associated (was a reconstruction of) a particle whose INDEX _I was "-5", than that track would have a ASSINDEX_I of "-5". That track's INDEX_I could, at the same time, be any positive integer.

## Grouping into categories and creation of histograms

For readers who have the package that we've been working with available, one of the files in the C++ package is labeled "stdafx.cpp". This file consists of a series of classes which, when given a particle or track as an argument, return ones or zeros that denote whether that particle or track is findable, acceptable, or, for the particles, whether its associated track is acceptable. This file is used as a utility file: the individual methods within it are called by other files that create histograms and fake track txt files. Essentially, the stdafx.cpp file is used by files as grouping filters which will create the end product; the histograms that the physicist will need to look at.

The stdafx.cpp file itself calls on a stdafx header file, wherein many of the parameters listed above are given minima and maxima. The stdafx.cpp file then uses these in a filtering processes that make the final decisions on whether individual particles are findable or tracks are acceptable. Following is a list of the final cuts that we made when sorting the particles and tracks into "findable" and "acceptable" categories, respectively. This is copied directly from the C++ code itself.

```
//findable particle cuts
#define RORG_MAX 20.0
#define RORG_MIN -1.0
#define COSTHETA_MAX 1.1
#define PATHLENGTH_MIN 750.0
#define PATHLENGTH_MAX 99999.0
#define PT_MIN 0.75
#define ALPHA_MIN -1.0
#define ALPHA_MAX 99999.0

//acceptable track cuts
```

```
#define T_DCA_MAX 100.0

#define T_COSTHETA_MAX 1.1

#define T_PATHLENGTH_MIN 0.0

#define T_PT_MIN 0.5
```

The file that does most of the work in creating the different end results of the code is labeled as the "Hist_Gen.cpp" file. This is the file that creates histograms, calls on the stdafx classes, and fills in the histograms. We will now list and detail the different "end product" histograms.

The first histograms are distributions of findable particles, and then of acceptable tracks, as functions of various parameters. For example, the histogram that is labeled as "DISTparticle_omega", will show a histogram whose x axis represents a range of different omegas, with a range from -0.005 to +0.005 ($cm^{-1}$). The Hist_Gen.cpp file will split this axis up into a number of bins, and in each bin it will fill a particle. The cut that is used for the entire histogram is a findable particle cut. Therefore, the only particles that show up on this histogram are findable particles. The histogram, then, shows a distribution over the parameter omega of findable particles. Physicists using the code can look at this spread and see which values of omega are most frequently held by the particles, which are least, etc. There is one of these for all of parameters which detail the particle (Cos$\theta$, alpha, path length, etc.).

In this group as well are histograms that are the same in everything as the above, except that the cut that is made is for acceptable tracks, not findable particles. Thus, these histograms will show distributions, spread over the same parameters, of all the acceptable tracks analyzed by the package.

Another group of histograms are labeled as "efficiency histograms". As for the first group discussed above, these histograms will as well only show findable particles. These findable particles are spread over the x axis in the same manner (according to a parameter, such as omega), but in these histograms, when a particle is filled, the Hist_Gen.cpp file checks to see if the particle has a tracks associated with it, and then if this associated track is acceptable, according to the cuts set up in the stdafx header. If the particle does have such a track, then a “1” filled into the histogram, at the appropriate x-axis position. If the particle does not have such a track associated with it, then a “0” is filled into the appropriate position. These increments are binned, as with the distributions discussed above, and then once a number of “ones” and “zeros” have been filled into the bin, the average value is taken over the bin. These histograms best show the efficacy of the track reconstruction code, whose goal it is to construct an acceptable track for every particle that moves through the detector. Thus, if the ratings on this histogram are lower at specific positions of the parameter, then the efficiency of the detector is lower at these positions of that parameter. For example, in the Cos (theta) efficiency histogram, we’d ideally see (in a world where everything works as it’s supposed to), a fall off of efficiency at higher values of $cos\theta$, where the particles are running closer to the original particle beam, and the efficiency of the reconstruction package ought to be lower, since it’s impossible to build instrumentation right along the beam line.

Before explaining the third finished product of the package, the fake track text files, we must explain the meaning behind the purity rating, PURTIY_I. When the JAVA package runs on the original raw data and reconstructs the tracks from the particle hit information, it lists in the flat file a value which denotes the “purity” of each acceptable track. As stated above, the tracks themselves are constructed from a series of hits which the particle has left behind as it zips through the detector. Because we are running simulations, and we have access to this “God-given” particle information, we can compare all these hits which create the track to the actual particle information that the track is supposed to represent. A track which has a “purity” rating of 1.0 has a list of hits for which every hit came from the particle that the track is associated with, and that the reconstructed track is meant to represent. If one of the hits on this track actually came from a different particle than the track’s associated one, than the track is impure, and the purity rating, using the simple equation (accurate hits/total hits), denotes to what extend it is pure. Within the C++ package, the programmer can input a minimum purity rating, PURITY_MIN, which will tells the program at which point in terms of purity the reconstructed track is labeled as a "fake track". For example, I set the PURITY_MIN to 0.85, so every track with a purity rating that is less than 0.85 will be labeled as a fake track.

The third finished product of the C++ code than are called the fake track text files. These are similar to the histograms above, they show a distribution over a varying parameter, but here there is no picture, only numbers and percentages. These list the number of fake tracks that are found in each small range of the parameter. As noted above, a fake track is an acceptable track that has too many hits that did not come from the same particle – some of its hits are actually associated with other particle(s). The text file shows how many fake tracks are in each bin, and what percentage of the total tracks these fake tracks represent.

## Results

Most of our time spent working on this project has gone into adapting the various packages of code that we have been working with. At the time of writing, we have finally been able to successfully produce our first results, to go from a raw data file, through the reconstruction, through the C++, and finally to histograms which read as we expect them to. We have found some interesting results already. These are discussed in what follows.

We start by looking at the results from a Zpole raw data file. These events have an energy of roughly 92 GeV, lower than the maximum 500GeV expected for the ILC. Thus, the jets will not be as densely collimated, and track reconstruction should be easier. The raw data will first be cut in the JAVA package so that the only events we view will have a thrust access with a cos$\theta_T$ between 0.6 and -0.6. In other words, the primary “jets” that emerge from the collision at the center of the detector will for this data come out at a theta angle between these values. Below we look at the histograms which show the reconstruction efficiency versus the track’s cos$\theta$ (Figure 1), the efficiency versus the alpha angle ($\alpha$) (Figure 2), and the efficiency versus the $p_T$ (Figure 3). We can see that the reconstruction package, with the cuts that we’ve made in the C++ coding, is very efficient. All along the range of cos$\theta$ there is no appreciable drop in efficiency. A similar thing can be seen with respect to the angle $\alpha$. The efficiency drops off sharply at low levels of $p_T$. This is commented upon in the summary below.

Figure 1: Eff vs $cos\theta$ for Zpole data, -0.6<$|cos\theta|$<0.6

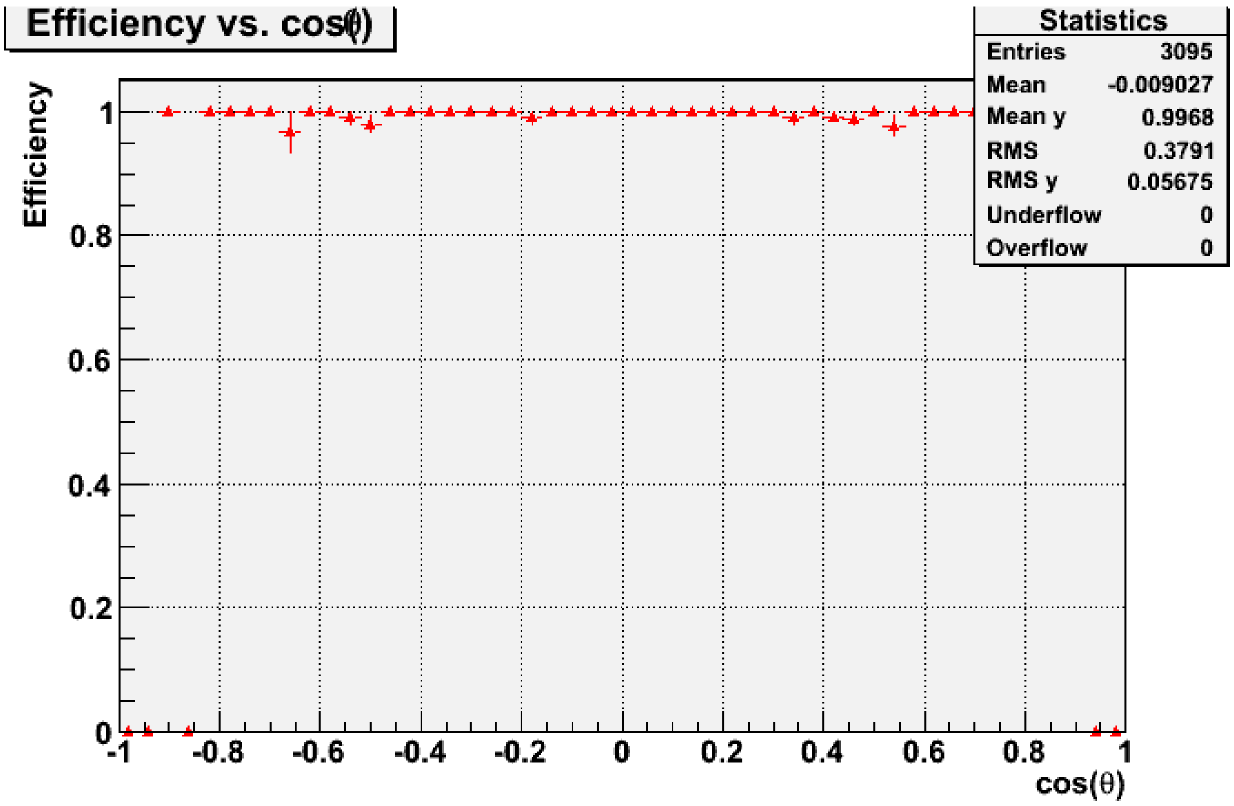


Figure 2: Eff vs $\alpha$ for Zpole data, -0.6<$|cos\theta|$<0.6

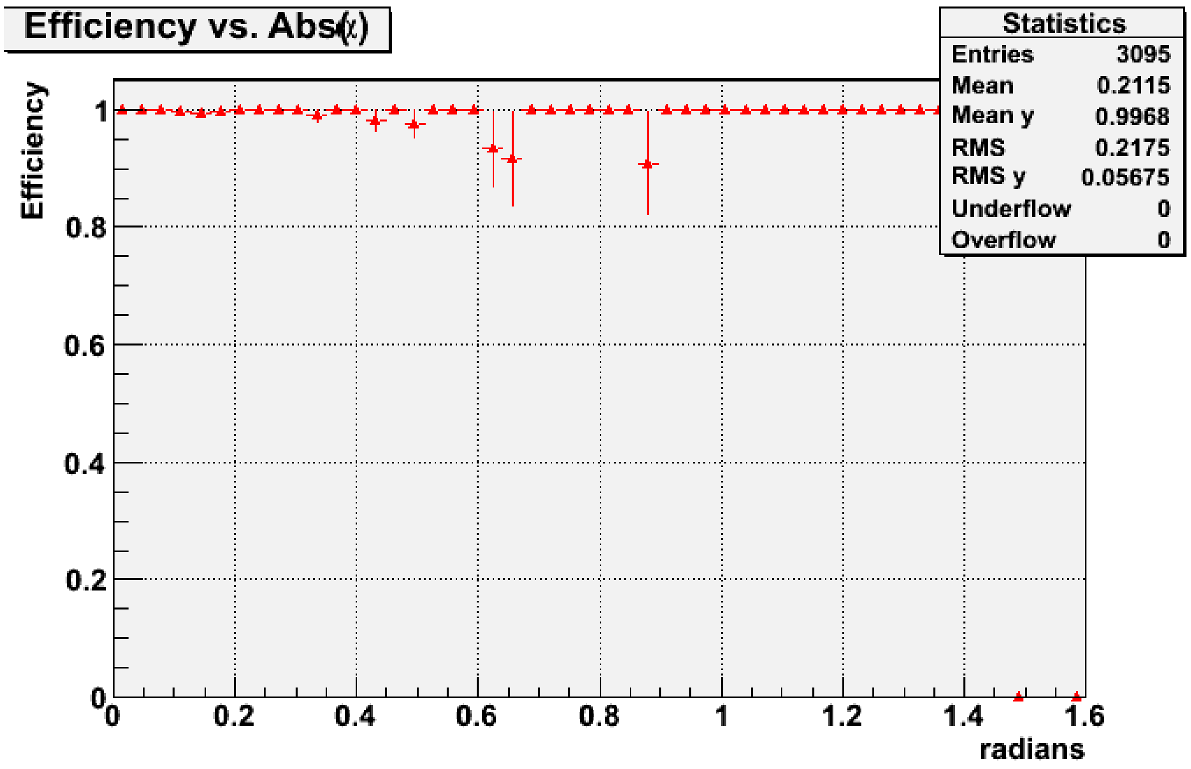
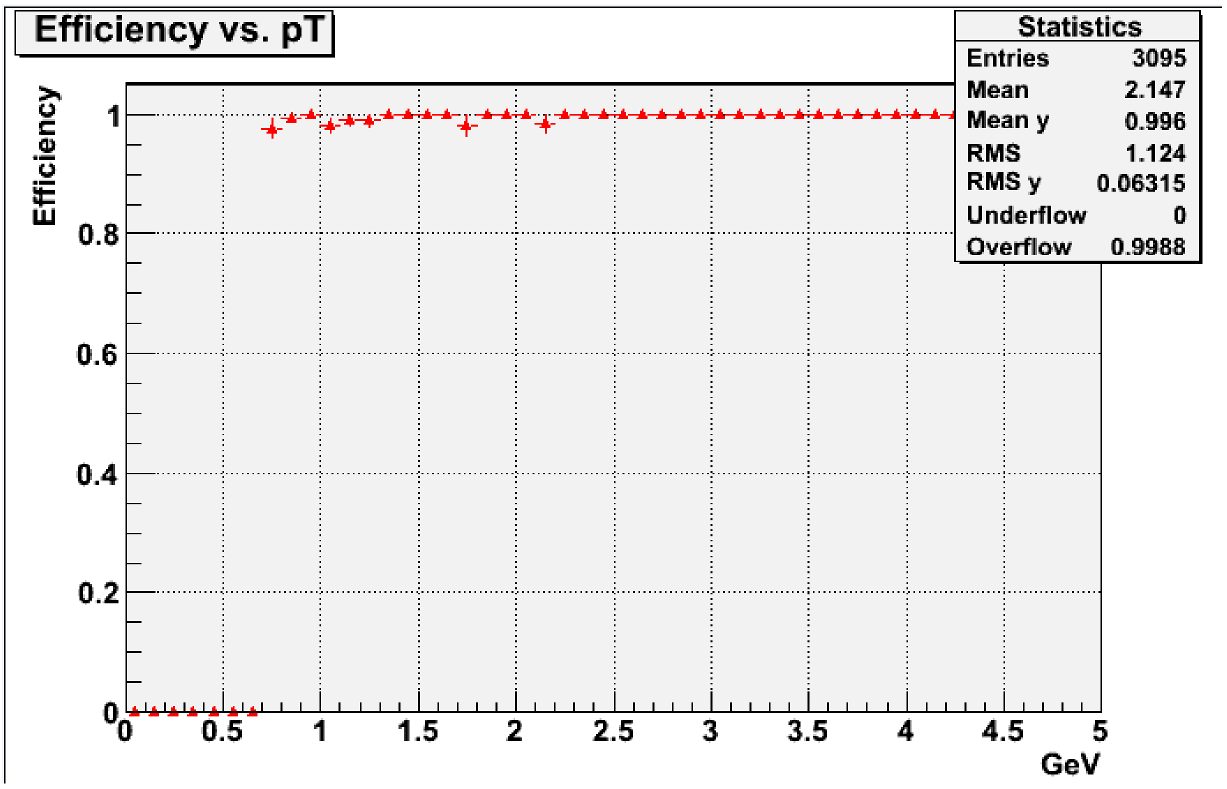


Figure 3: Eff vs $p_T$ for Zpole data, -0.6<$|cos\theta|$<0.6



Next we look at the same Zpole raw data, only this time we make the cos$\theta$ cuts on the event jet at -0.95 and 0.95. The efficiency is consistently high in these histograms as well. They look roughly identical to what we see with the 0.6 cuts. The reconstruction algorithm is remarkably successful at reconstructing tracks close to the beamline.

Figure 4: Eff vs $cos\theta$ for Zpole data, -0.95<$|cos\theta|$<0.95

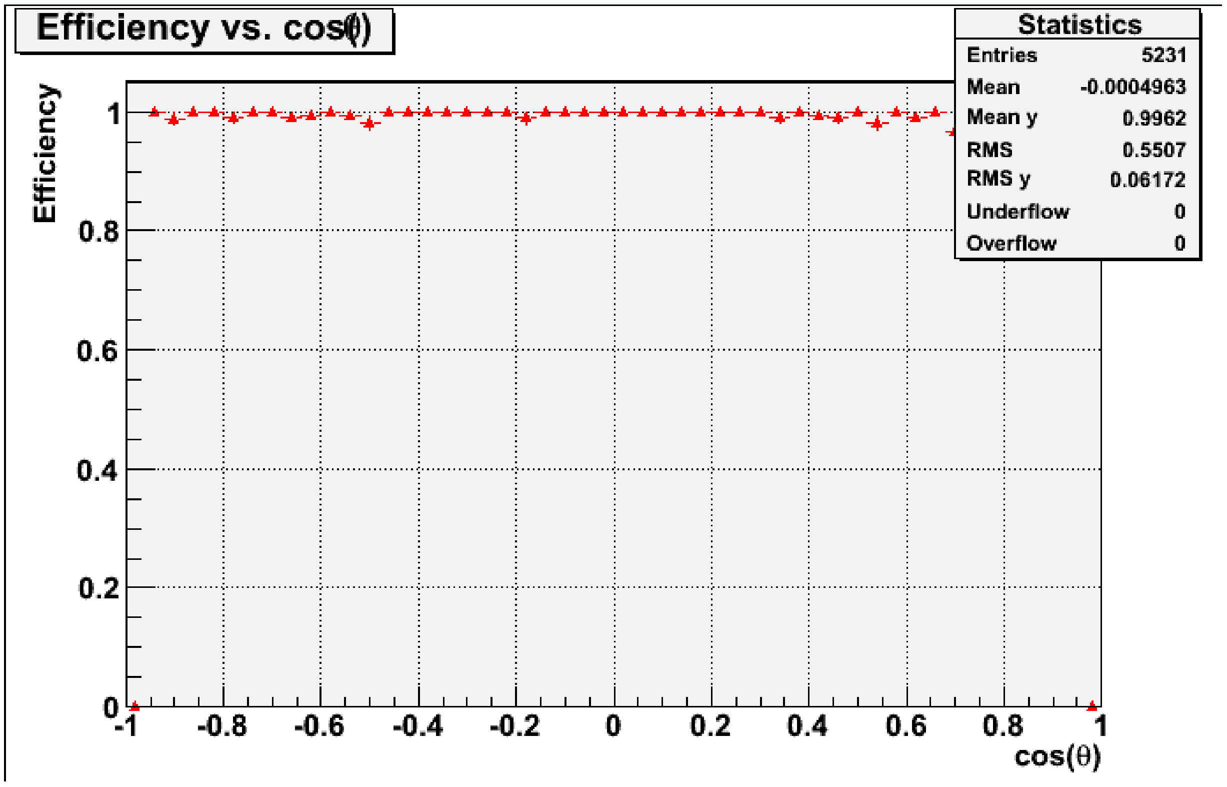


Figure 5: Eff vs $\alpha$ for Zpole data, -0.95<$|cos\theta|$<0.95

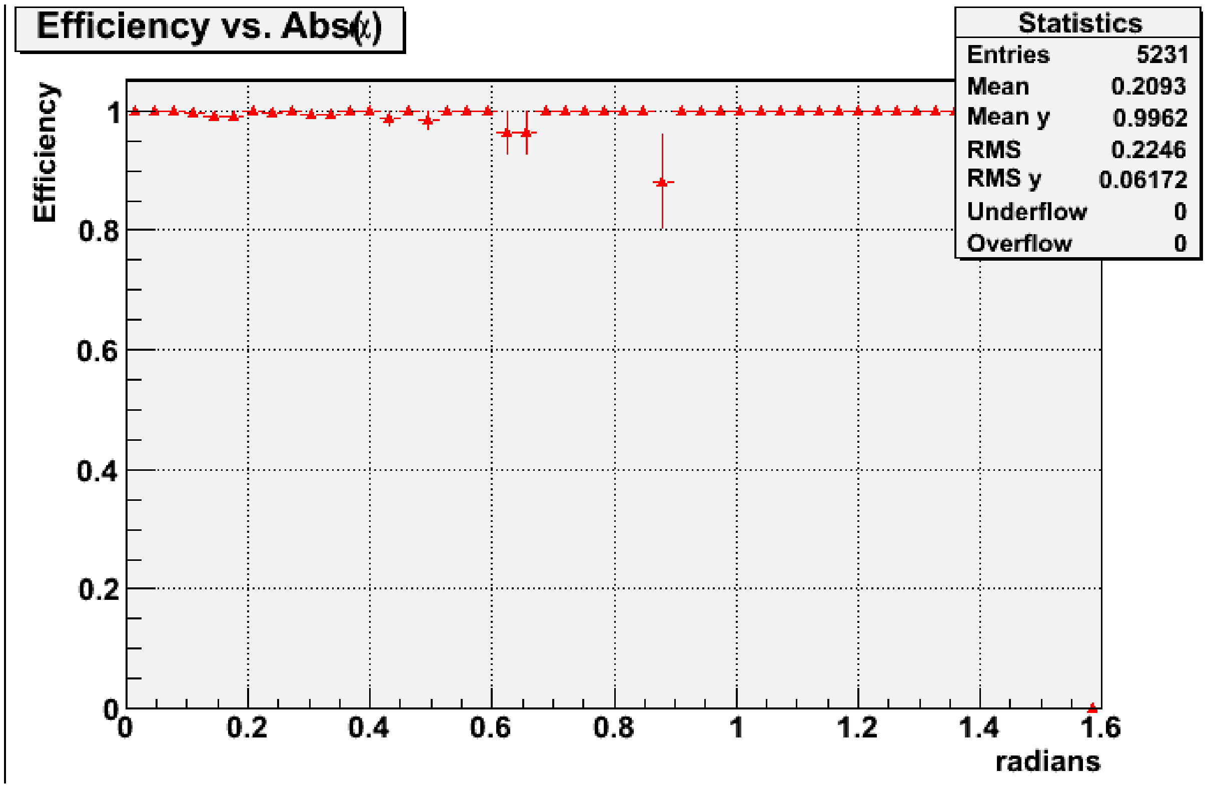
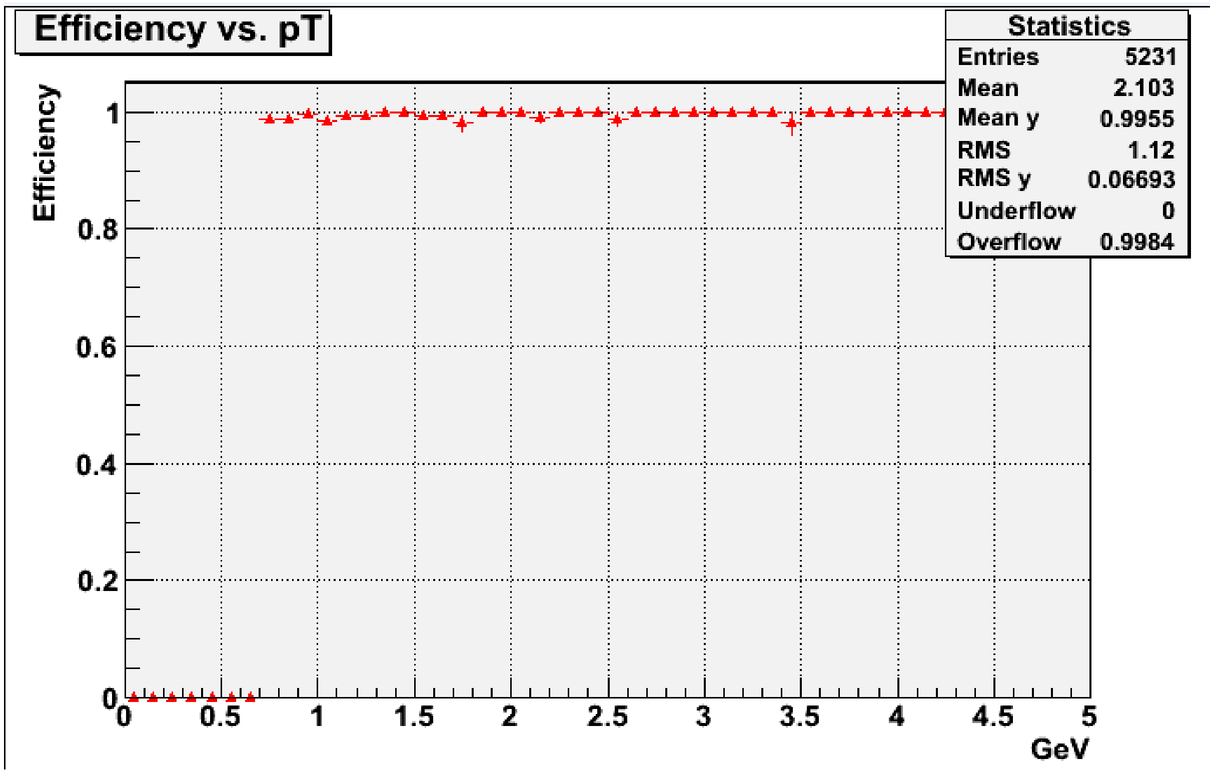


Figure 6: Eff vs $p_T$ for Zpole data, -0.95<$|cos\theta|$<0.95



We then move on to “pythia” raw data files. These show events that occur at 500 GeV and have jets that are much more densely collimated, potentially confusing attempts to reconstruct their tracks. The first (Figures 7-8) are the events that have jets whose cos$\theta_T$ fall between -0.6 and 0.6, as with the Zpole data above. Looking at the efficiency vs $\alpha$ (Figure 7), we can see that there is a small ($\approx 2\%$) dip in efficiency near the core of the jet, for which at $\alpha = 0$. It appears that the reconstruction algorithm is indeed having some trouble in the core of these dense jets.

Figure 7: Eff vs $\alpha$ for pythia data, -0.6<$|cos\theta|$<0.6

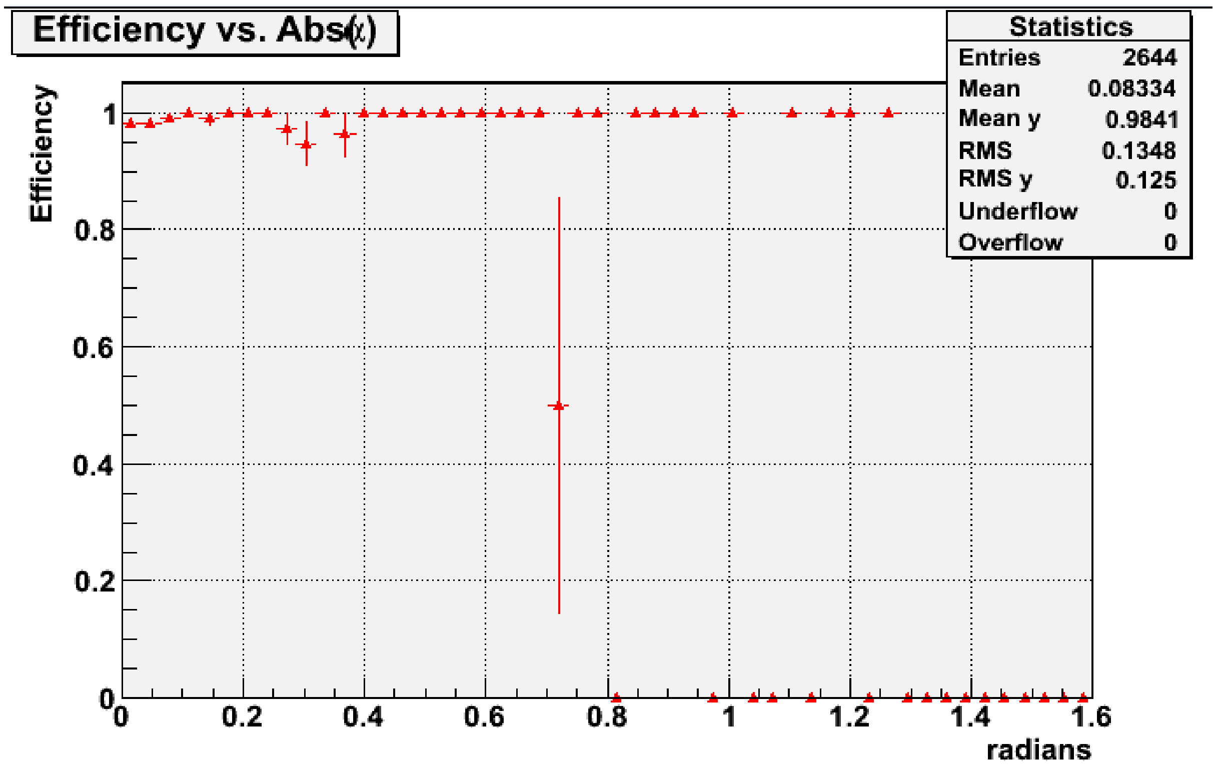
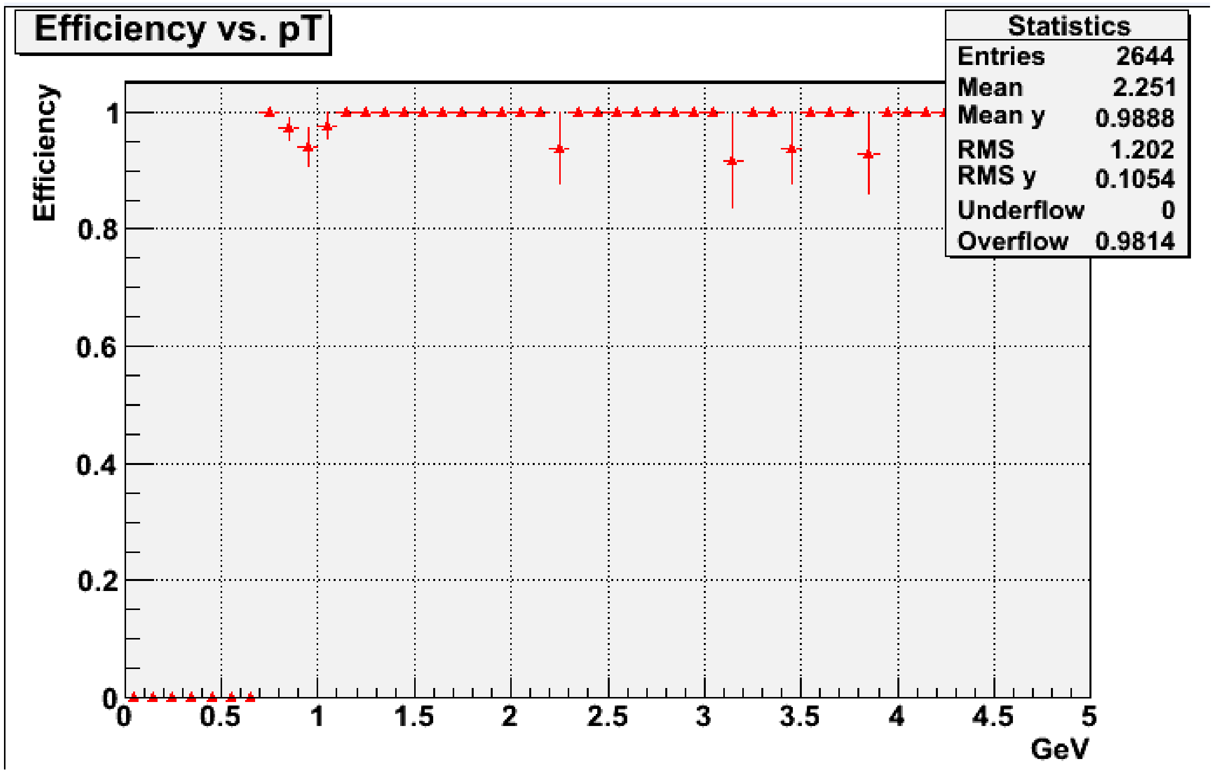


Figure 8: Eff vs $p_T$ for pythia data, -0.6<$|cos\theta|$<0.6



| Statistics | |
|---|---|
| Entries | 2644 |
| Mean | 2.251 |
| Mean y | 0.9888 |
| RMS | 1.202 |
| RMS y | 0.1054 |
| Underflow | 0 |
| Overflow | 0.9814 |

In the pythia data that has event cuts at -0.95 and 0.95, we see a similar dip in alpha.

Figure 9: Eff vs $\alpha$ for pythia data, $-0.95<|cos\theta|<0.95$

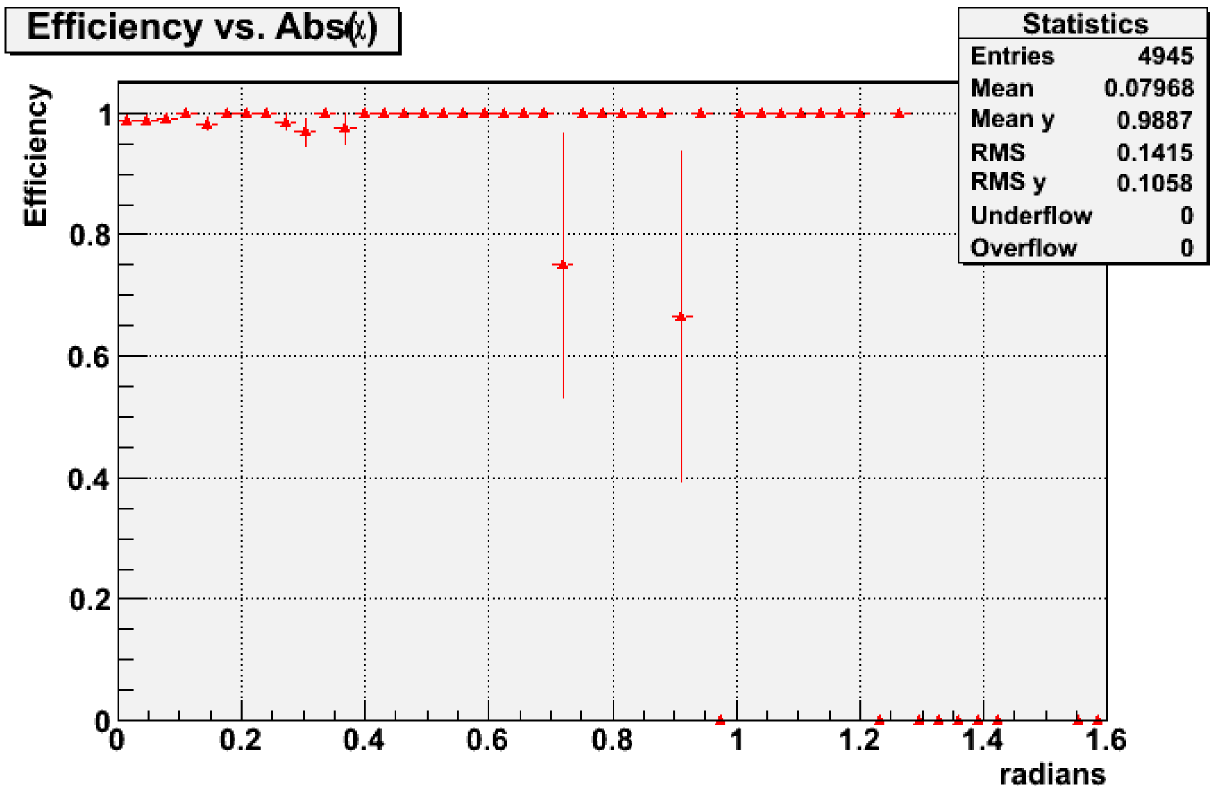
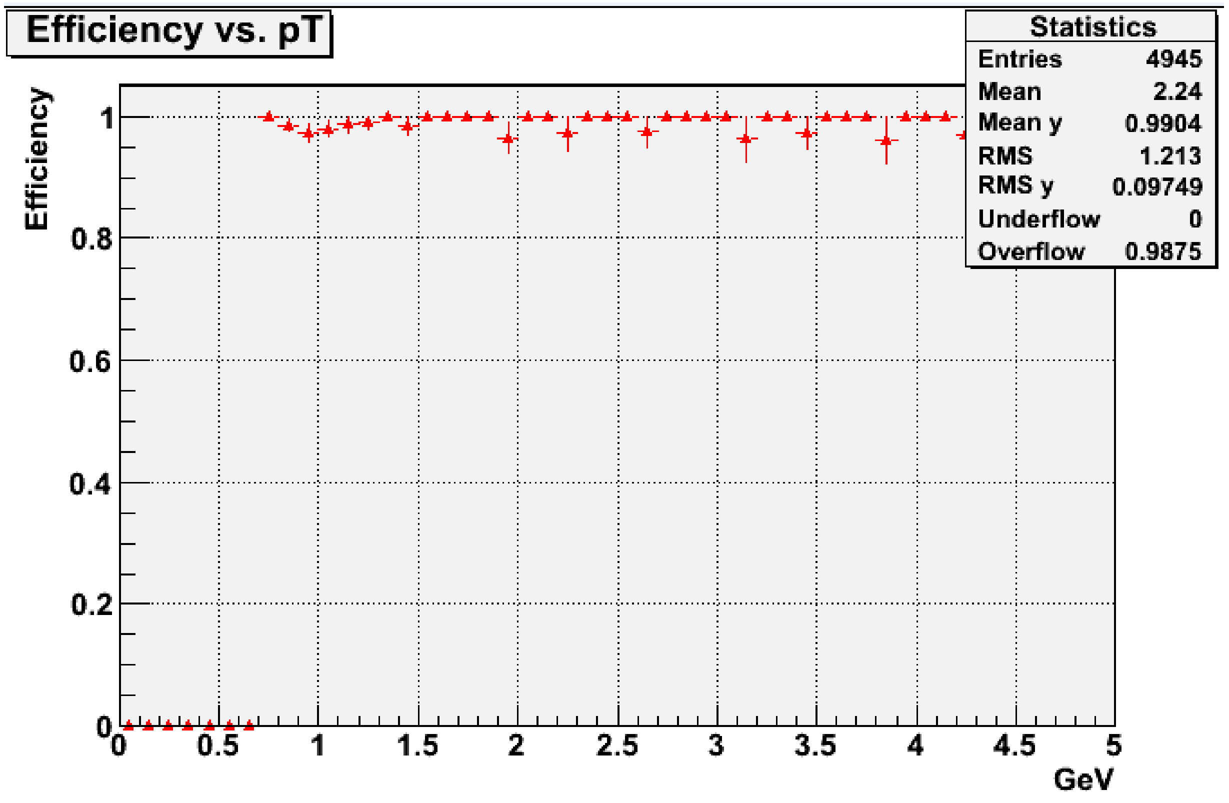


Figure 9: Eff vs $p_T$ for pythia data, -0.95<$|cos\theta|$<0.95



Below are a few more examples of data that our C++ package prints out. We have taken this particular set from the pythia raw data with a cos$\theta$ cut of 0.6.

Figure 10: Eff vs path length for pythia data, -0.6<$|cos\theta|$<0.6, where we have eliminated cuts for findable particles in our C++ code on the ranges of $p_T$ and path length.

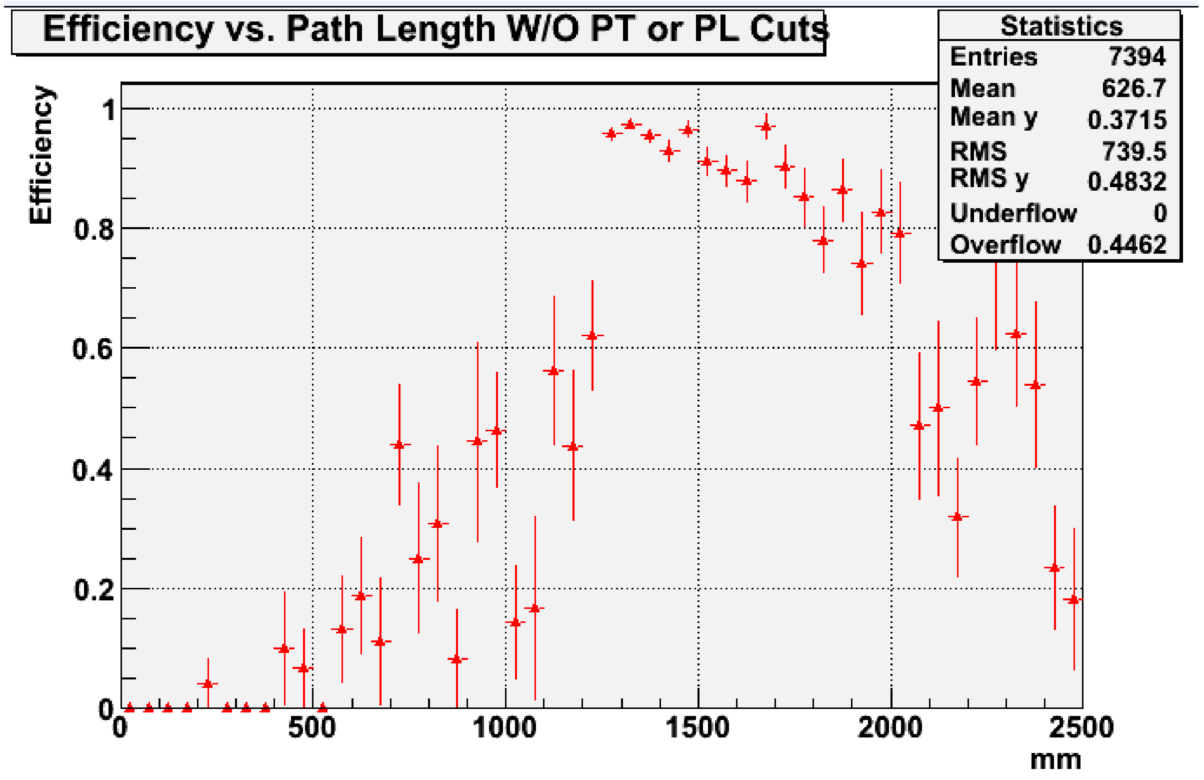


Figure 11: Eff vs $p_T$ for pythia data, -0.6<$|cos\theta|$<0.6, where we have eliminated cuts for findable particles in our C++ code on the ranges of $p_T$ and path length.

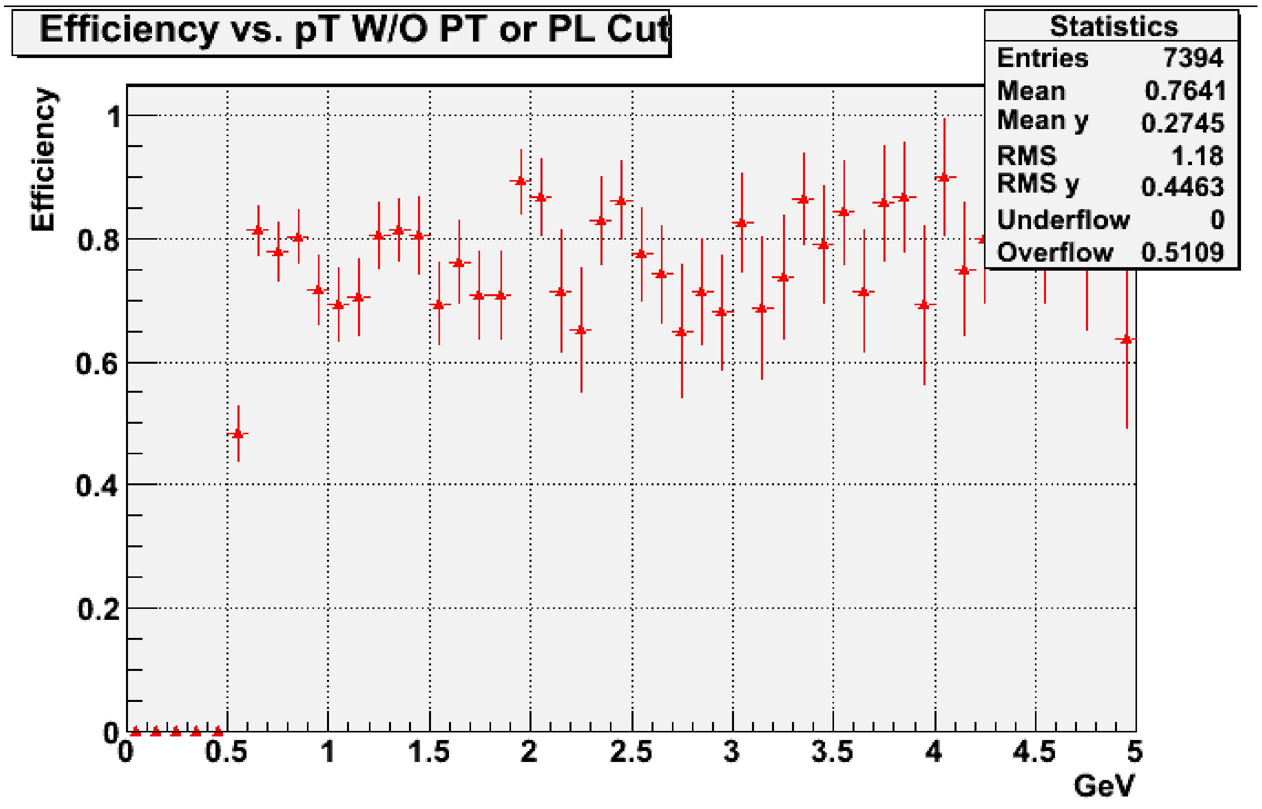
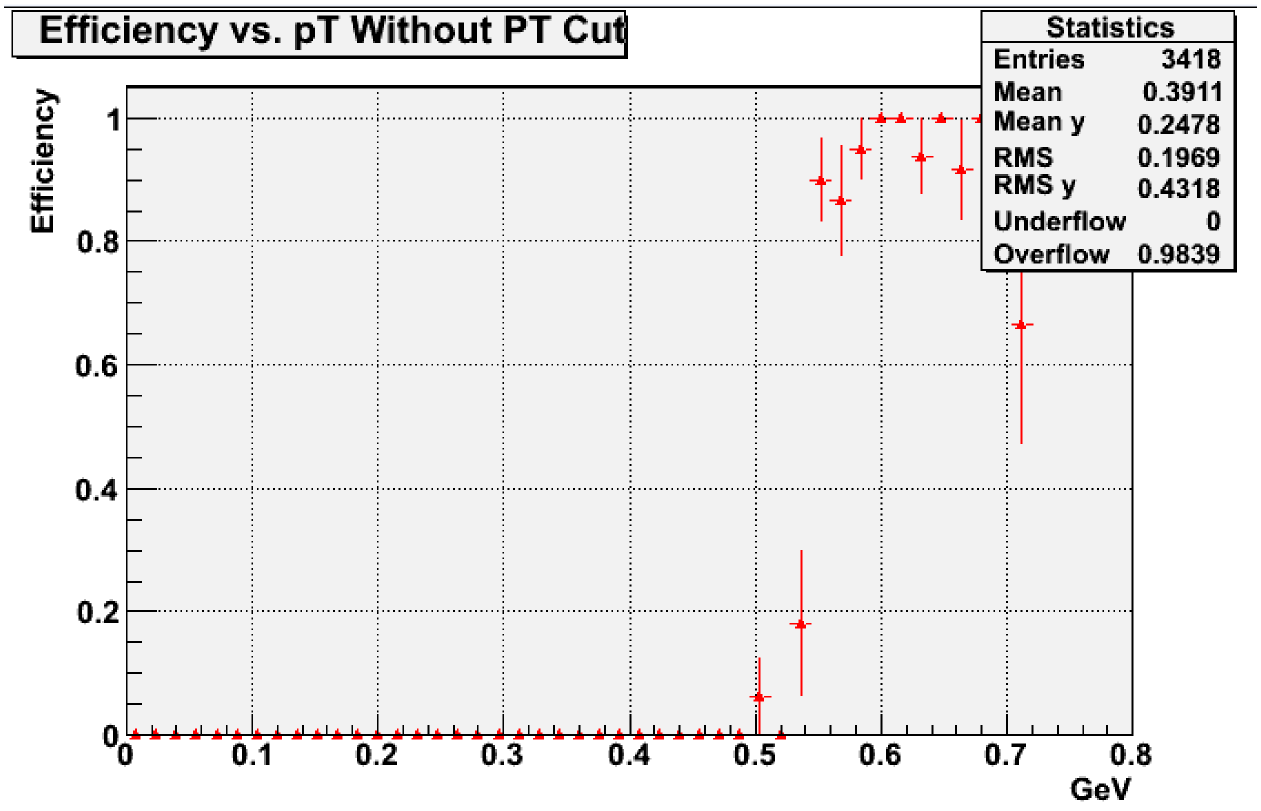


Figure 12: Eff vs $p_T$ for pythia data, -0.6<$|cos\theta|$<0.6, where we have eliminated cuts for findable particles in our C++ code on the range of $p_T$, and where we have adjusted the evaluated range.



## Summary

In conclusion, we have adapted the track reconstruction code to run on the latest framework, updated the C++ code to display efficiency histograms that are more in depth and easier to analyze, and corrected for bugs. We can see from the data that the reconstruction yields very good efficiency ratings at high values of cos$\theta$. We also observe a fall in efficiency at low levels of $p_T$. This can be expected from the geometry of the detector, since at low momentum the particle will spiral in a very tight helix and it won't be able to cross the outer detection layers of the detector. Finally we see a slight drop in efficiency in the core of the jet when we analyze the higher energy (500GeV) events. This indicated that track reconstruction is becoming difficult in the core of jets, since that for these events the jet is very densely collimated, causing more confusion in the core of the jet.

## Acknowledgements

I would like to acknowledge Dr. Bruce Schumm, my advisor, for his endless patience with me throughout this very long process and for his telling of many interesting jokes. I would also like to acknowledge Chris Meyer, who in the previous academic year had written most of the code that I’ve been dealing with throughout this one and who, while he’s attending graduate school in Chicago, still finds time to help us all out, for no other reason than to just help out. He’s also a genius. I would as well like to thank Chris Betancourt, Brian Colby, and John Willey for answering random computer management questions that I’ve had.

## References

[1] McCormick, Jeremy, “Getting Started with org.lcsim”.

<http://confluence.slac.stanford.edu/display/ilc/lcsim+Getting+Started> Accessed 08/08 – 02/09.

Used as tutorial and general reference.  This index site is part of a large group of sites found at

<http://www.lcsim.org/>.  This page was often used a s a starting point in looking for information.

[2] Various authors, “ILC Home”.<http://www.linearcollider.org/cms/> Accessed 01/09 – 02/09.

This site was used as a reference for general backround information.

[3] Soulie, Juan, “C++ Language Tutorial”. <http://www.cplusplus.com/doc/tutorial/> Accessed 08/08 – 02/09.

This is a very logical and concise tutorial, used as such and as a reference during the debugging process.

[4] Hommel, S., Zakhour, S., Royal, J., Rabinovich, I., Risser, T., Hoeber, M., “The JAVA Tutorial: A Short Course on the Basics, 4th Edition”. 2006. Accessed at <http://JAVA.sun.com/docs/books/tutorial/> Accessed 11/08 – 02/09.

This is a general JAVA tutorial, used as a reference for the basics of the language.