OBJECT-ORIENTED ANALYSIS CODE FOR HALL A VERTICAL DRIFT CHAMBERS

Jonathan Robbins[A] and Jens-Ole Hansen, Ph.D.[B]

ABSTRACT

The high-resolution spectrometers in Jefferson Lab's Hall A use vertical drift chambers to determine charged particle tracks. The current analysis code for the vertical drift chambers is difficult to maintain and modify, which has prompted the development of an object-oriented version, which will be easier to maintain and more able to adapt to changes in the detector configuration. However, the object-oriented approach involves using a slightly different algorithm than ESPACE, which could lead to different results. In this project, a preliminary version of an object-oriented analysis program for the vertical drift chambers is created and its results are compared to the existing software to determine the impacts of the differences in the reconstruction algorithms. In addition, the algorithms themselves are compared, and minor differences in track reconstruction techniques are reported.

INTRODUCTION

Jefferson Lab is one of the few nuclear physics research facilities in the world to feature a continuous electron beam, which allows unique opportunities to study subatomic structure. The largest of Jefferson Lab's three experimental halls, Hall A, contains two High Resolution Spectrometers (HRSs), which can be used to make precision measurements of the momentum and origin of scattered charged particles. Each of the HRSs has a set of Vertical Drift Chambers (VDCs), which are used to detect charged particle tracks in the focal plane of the spectrometer. The software used to analyze the data collected by the VDCs and extract momentum and position information of particles is discussed in this paper.

Until now, Hall A has been using a software package called ESPACE (Offermann, 1997) for track reconstruction. ESPACE is written in FORTRAN and, for various technical reasons, is difficult to maintain and to modify when changes in the detector configuration are necessary. As a result, an effort has been started to duplicate the capabilities of ESPACE using object-oriented programming. This project is called the C++ Analyzer Project. By using object-oriented techniques, the analysis code is expected to become more maintainable and flexible. The C++ analyzer is based on ROOT (Brun & Rademakers, 2001), an object-oriented data analysis framework that has been developed at CERN since 1995 and that is specialized for physics applications.

The goal of this project is to determine the effect of small differences between the C++ analyzer algorithm and the ESPACE algorithm. The C++ analyzer's object-oriented design encourages encapsulation of data and processing at the object level, while ESPACE uses global processing and does not strongly distinguish separate components of the program. Also, the C++ analyzer was designed to break the analysis into separate coarse and fine tracking stages, rather than having a single stage, like ESPACE.

MATERIALS AND METHODS

The code for the object-oriented analyzer was written in C++ and compiled using the GNU C++ compiler (gcc/g++ 2.91) on a Pentium III 650 MHz computer running RedHat Linux 6.2. ROOT was used to view the results of the object-oriented code. ESPACE was used to analyze the data as a reference, and Paw++ (CERN, 2001) was used to view the results from ESPACE. The input data were obtained as part of the optics study for Hall A experiment E97-111 in September 2000. The data analyzed here were taken with the left arm HRS, using a 9-foil graphite target. The target foils are positioned perpendicular to the beam, and the distance between the foils is 3.988 cm. A sieve slit with 7x7 holes was used to block portions of the beam at the spectrometer entrance, which produces peaks in the angular spectra of the data.

As mentioned earlier, the C++ analyzer uses an object-oriented approach to implement data analysis functions. One of the first steps in the design process of the VDC detector code, therefore, was to define a set of suitable C++ classes, their features, public interfaces, and their interactions with each other. The classes, which are described in detail below, were chosen so as to closely correspond to the physical components of the VDCs and objects relevant to the analysis.

Once the class design was in place, the next step was writing a prototype so that the program could successfully go through each step in the analysis process. Once this was completed, the more challenging task of refining the analysis code began. Initially, it was sufficient to find results that were clearly wrong based on physical reasons and to track down the causes of these errors. Next, ESPACE was used to analyze the same data as the C++ analyzer to look for major differences in the results, which would indicate serious errors in the C++ algorithm. Finally, once there was sufficient agreement between the ESPACE and C++ analyzer results, subtle differences were investigated.

A: University of Richmond, Richmond, VA; B: Thomas Jefferson National Accelerator Facility, Newport News, VA

Description of Vertical Drift Chambers

The VDCs are the most sophisticated detectors in the spectrometers, since they are designed to provide sub-millimeter track position resolution. The Hall A VDC packages consist of 4 gas-filled wire chambers with 368 active wires each (Leathers, 1996; Fissum, 2001). The gas is a mixture of 50% argon and 50% ethane by volume (62% and 38% by mass). The wire chambers are arranged into an upper VDC and a lower VDC, each of which consists of two wire planes whose wires are oriented at 90° to each other. The wire planes are tilted at 45° to the nominal beam direction such that an average of 5 wires receive a signal for each event. The convention used in this paper is that the direction perpendicular to the wires in the first and third wire planes (the "U" wire planes) is the *u* direction and the direction perpendicular to the wires in the second and fourth wire planes (the "V" wire planes) is the *v* direction.

When a particle passes through a wire chamber, it creates positively charged ions and free electrons. The wires are kept at ground potential and negatively charged high voltage plates are located above and below the wires, so that the electrons are attracted to the wires (Liyanage, 1999; Wechsler, 1996). As the electrons move, they accelerate and collide with other gas molecules which ionizes them. When this process occurs near the large electric field gradients created by the wires, the result is an avalanche of electrons that strikes the wire, inducing a signal (or "hit") on that wire. The signal is then preamplified and discriminated before reaching a time-to-digital converter (TDC). The TDCs measure the time elapsed between the signal's arrival and the arrival of a common stop signal, triggered by the particle passing through a scintillator panel. As a result, the time resolution of the TDCs, 0.5 ns, determines the time resolution of the VDCs. The time measured by the TDCs can be converted into the so-called drift time, i.e. the time taken for the electrons freed by the charged particles to drift to a wire. The drift time information can be used to determine the position and direction of the particle crossing each wire plane. This information, in turn, allows reconstruction of the particle's track.

Track Reconstruction Algorithm

General Track Reconstruction

Track reconstruction for both ESPACE and the C++ analyzer involves the following basic steps. First, hits on adjacent wires, presumably caused by the same particle, are grouped into a "cluster." Next, clusters found in each of the 4 wire planes are matched, and the position at which the track crossed each wire plane is determined. To find this position, it is first necessary to apply an algorithm to convert the drift times measured by the TDCs into the drift distance of the electrons, where the drift distance is defined to be the distance perpendicular to the wire plane from a wire to the particle track. To do this conversion accurately, corrections for the angle of the track and the nonuniformity of the electric field near the wires must be included (Wechsler, 1996). Once drift distances have been calculated, a linear fit is used to determine where the track crossed each wire plane, and the position and direction of the particle when it crossed the focal plane are calculated.

The software uses the *u*-*v* and detector coordinate systems (Offerman, 1997) before producing a final result for the position and direction of the tracks in the TRANSPORT (Brown et al., 1983) coordinate system. In this system, z_T is the nominal direction of the beam, x_T is the direction the beam moves when its momentum is increased an infinitesimal amount, and y_T is the direction appropriate to make a right-handed coordinate system (Figure 1). The angle θ_T is measured from the z-axis to the projection of the track in the *xz*-plane, and the angle ϕ_T is measured from the *z*-axis to the projection of the track in the *yz*-plane. Straightforward trigonometry gives the following equations, used to convert from the *u*-*v* coordinate system (u, v, θ_u, θ_v) to TRANSPORT coordinates (x_T, y_T, θ_T, ϕ_T):

$$\begin{aligned}
x_T &= \frac{u\sin\phi_v - v\sin\phi_u}{\sin(\phi_v-\phi_u)}\cos\rho + z_D\sin\rho \\
y_T &= \frac{v\cos\phi_u - u\cos\phi_v}{\sin(\phi_v-\phi_u)} \\
\tan(\rho+\theta_T) &= \frac{\tan\theta_u\sin\phi_v - \tan\theta_v\sin\phi_u}{\sin(\phi_v-\phi_u)} \\
\tan\phi_T &= \frac{\tan\theta_v\cos\phi_u - \tan\theta_u\cos\phi_v}{\sin(\phi_v-\phi_u)\cos\rho + (\tan\theta_u\sin\phi_v - \tan\theta_v\sin\phi_u)\sin\rho}
\end{aligned} \qquad (1)$$

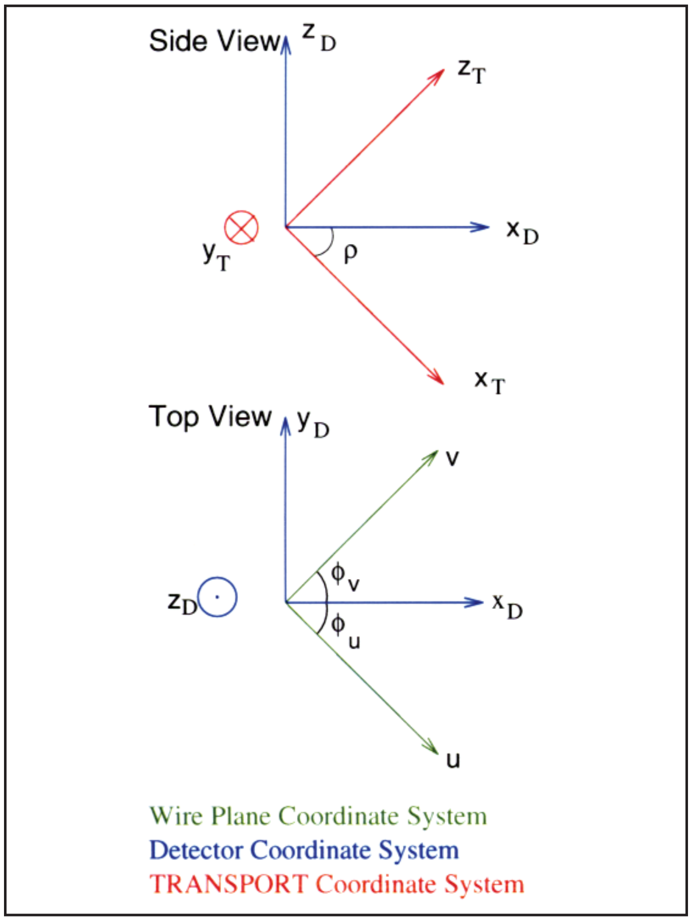
Side View
z_D
z_T
y_T
x_D
ρ
x_T
Top View
y_D
v
z_D
ϕ_v
x_D
ϕ_u
u
Wire Plane Coordinate System
Detector Coordinate System
TRANSPORT Coordinate System

Figure 1. The TRANSPORT coordinate system used to reconstruct tracks in the C++ analysis code.

In the previous equations, ϕ_u and ϕ_v are the angles that the u and v axes make with the projection of the x direction into the uv-plane, ρ is the angle between the detector plane and the z_T axis, z_D is the distance from the focal plane of the spectrometer to the U wire plane in which the track is being reconstructed, and θ_u and θ_v are the angles that the track makes with the u and v axes. In addition, the TRANSPORT coordinates are then projected into the TRANSPORT plane ($z_T = 0$) using

$$\begin{aligned} x'_T &= x_T(1+\tan\rho\tan\theta_T) \\ y'_T &= y_T + \tan\rho\tan\phi_T x_T \end{aligned} \qquad (2)$$

where x'_T and y'_T are the projected coordinates of the track.

Object-oriented Track Reconstruction

This section describes in detail the C++ analyzer code to convert the raw drift times into actual track information. The first stage is decoding the raw data for each event into hits in a specific wire plane with their associated drift times and wire numbers. The second stage is the "coarse tracking" stage, in which a rough estimate of the particle's track is determined. This stage is particularly useful for situations when a quick estimate of results may be desirable. The third stage is the "fine tracking" stage. During fine tracking, the position and direction of the particle track found in the coarse tracking stage is refined.

During the coarse tracking, the hits are grouped into clusters by identifying hits with no more than one wire separating them. Once clusters have been found, clusters in the U wire plane are paired with clusters in the V wire plane by finding clusters whose hits with largest TDC values had drift times closest to each other. Equation 1 is then used to obtain an estimate of where the track found in the lower VDC would intersect with the upper VDC, and vice versa, and these estimates are used to match the clusters in the upper and lower VDCs. Once clusters from the upper and lower VDCs are matched, the global angles based on the cluster positions in each plane are calculated.

During the fine tracking stage, a drift-time-to-drift-distance (t-x) conversion algorithm is applied to the hits. Changing the t-x conversion algorithm can be accomplished very simply by using class inheritance. In principle, each wire could be associated with a different algorithm. The algorithm used in this analysis employs a polynomial in the angle of the track to provide a correction to the drift distance, where the angle of the track is obtained during the final step of the coarse tracking. Once the drift distance for each hit is known, a linear fit is carried out to calculate the point where the track crossed the wire plane (Figure 2). Equations 1 and 2 are applied again to match clusters from the lower and upper VDCs, and a final calculation of the track is made.

Class Description

The classes in the C++ analyzer closely correspond to the physical layout of the detector and to the logical structures that arise from the analysis algorithm. Figure 3 shows the class ownership hierarchy.

• *THaVDC*: The VDC class is the top class in the hierarchy, representing the entire VDC package of a single spectrometer. It contains data applicable to the entire VDC and two UV plane objects. It also contains functions responsible for the final calculation of track position and direction.

• *THaVDCUVPlane*: The UV plane class represents the upper and lower VDCs in the VDC package. Since each VDC has a U and V plane, the UV plane class has two plane objects in addition to other data specific to the VDCs. The UV plane class also generates UV track objects.

• *THaVDCPlane*: The plane class represents an individual wire plane in the VDC package. It contains an array of 368 wire objects, an array of hits, and an array of clusters. The size of the hit and cluster arrays vary with the actual number of hits and clusters. The plane class is responsible for decoding raw data into hits and finding clusters.

• *THaVDCUVTrack*: The UV track describes a track through one of the UV planes. UV tracks are able to search for a partner track in the other UV plane and are considered successfully matched when a track is its partner's partner.

• *THaVDCCluster*: The cluster class describes a cluster of hits. It has an array of pointers to the hits that belong to it. Also,

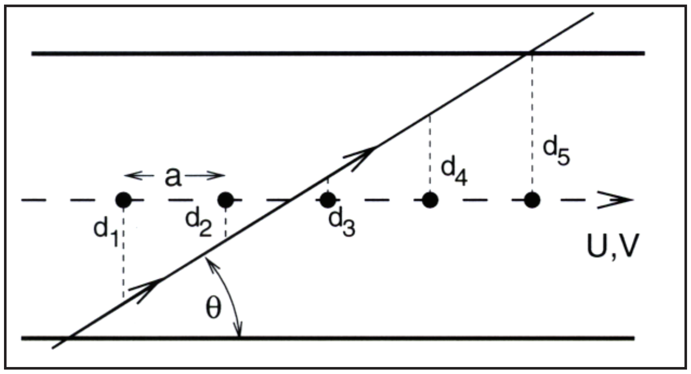

Figure 2. A particle track through a wire plane for a typical 5-wire event. The value d_i is the drift distances for ith wire in the cluster to be hit, a is the wire separation, and θ is the angle of the track with the wire plane.

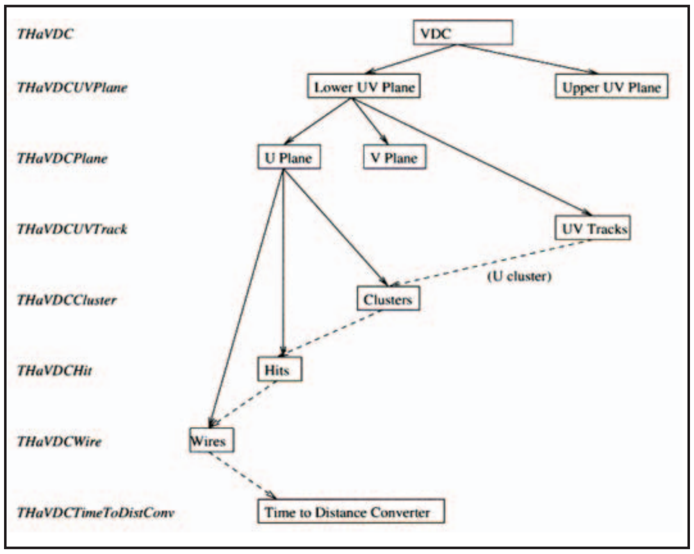

Figure 3. This diagram shows the class ownership hierarchy. The solid arrows indicate classes that are members of other classes. The dashed arrows indicate references to objects of other classes. For example, the VDC class has a lower UV plane object, and hits contain a reference to the wire on which they occurred.

it can perform a linear fit on the hits in order to estimate the point in the cluster where a track intercepts the wire plane and the angle of that track.

- *THaVDCHit*: The hit class represents a single hit on one of the wire planes. It contains the TDC value associated with the hit, the drift time of the electrons that caused the hit, and a pointer to the wire on which the hit occurred.
- *THaVDCWire*: A wire class object represents each wire in the VDC package. Each wire object contains the wire number and position of the wire in the detector. Also, each wire has a pointer to a time-to-distance conversion object for use by hits that occurred on the wire.
- *THaVDCTimeToDistConv*: The time-to-distance conversion class is a base class from which to derive actual *t-x* conversion algorithms. The main feature of the class is a function called `ConvertTimeToDist()`, which takes a drift time and track angle as arguments and returns a drift distance.

RESULTS

Three quantities that are of inherent interest for any VDC are the drift time distribution, the wire efficiency, and the intrinsic timing resolution. The shape of the drift time spectrum for a wire in a VDC is very distinctive. There is a large peak for short drift times due to the increased electric field intensity followed by a long plateau (Wechsler, 1996). Figure 4 shows a sample drift time spectrum for a wire from the VDC package. Other wires show very similar results. The efficiency of a wire can be determined by checking whether or not it fires when the two wires adjacent to it fire. Thus, the wire efficiency is defined by $\varepsilon = \kappa/(\kappa+\lambda)$, where κ is the number of times a wire fires when its neighbors fire and λ is the number of times it does not fire when its neighbors fire (Leathers, 1996). Figure 5 shows the wire efficiency for data with an even distribution of wire hits. The time resolution measures how precisely the firing times of events may be determined. The formula for timing resolution for an event with n hits is $\Delta T = |(t_1-t_2)-(t_n-t_{n-1})|$, where t_i is the time of the ith hit (Fissum et al., 2000). The time resolution as determined by the C++ analyzer is shown in Figure 6.

As mentioned, the data used to compare the ESPACE and C++ analyzer results were part of an optics study with a 9-foil carbon target. Results for both ESPACE and the C++ analyzer are shown in Figures 7-10. For this configuration, one expects a single narrow peak in the x_T direction because x_T depends mostly on the momentum of the scattered electrons, which is nearly constant here (elastic scattering from carbon). The clean peaks in the θ_T spectrum arise because the sieve slit collimator selects well-defined out-of-plane scattering angles at the target. Both y_T and ϕ_T depend in a complex way on the scattering position along the beam and the in-plane scattering angle at the target, and so one expects spectra with some structure due to the sieve slit pattern and target foils. The visible peaks are smeared because the results are projected into the TRANSPORT plane, rather than being in the focal plane of the spectrometer, and because the contributions from the target foils and the sieve slit are convoluted in these one-dimensional spectra. The mean x_T' value of the C++ analyzer results and ESPACE results differ by 2 mm, which is significantly greater than the detector resolution of 0.1 mm. Also, the shape of the x_T' spectra, as seen in Figure 7, is somewhat different,

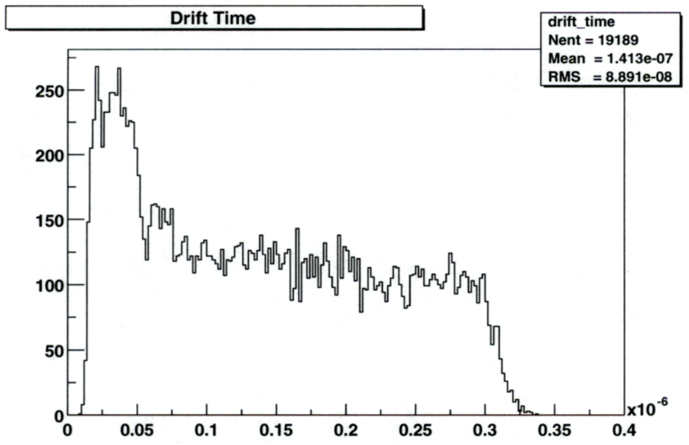

Figure 4. Drift time spectrum for wire 233 in the U_1 wire plane, calculated by the C++ analyzer. Note that that there is a peak for short drift times followed by a long plateau, as is expected. The x-axis is in seconds.

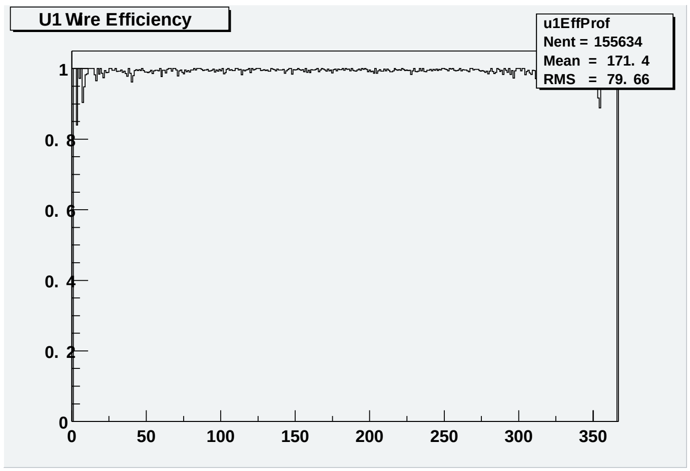

Figure 5. Wire efficiency in the U_1 wire plane, calculated by the C++ analyzer.

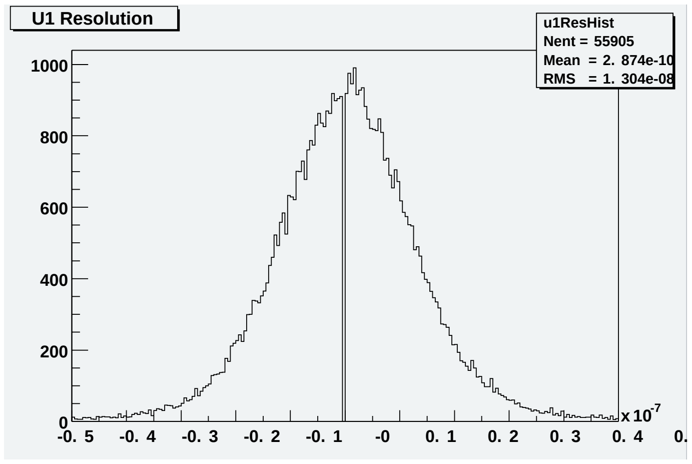

Figure 6. Time resolution for the U_1 wire plane, calculated by the C++ analyzer. The x-axis is in seconds.

especially on the right edge. Since both algorithms are operating on the exact same data, this difference is purely due to differences in the algorithms. The ESPACE results are likely to be more accurate, since ESPACE has been in use for many years, but the best way to determine the accuracy of the two systems would be to test them on theoretical data, where the correct answer is known. In the y_T' direction, 7 peaks are visible in the focal plane, but the projection of these values into the TRANSPORT plane (Figure 8) makes the peaks overlap significantly. The shape of the C++ analyzer and ESPACE results are very similar, though there is a small shift in the results, which may be due to the fact that x_T' and y_T' are coupled (Equation 2). The C++ analyzer results for θ_T and ϕ_T agree very well with the ESPACE results (Figures 9 & 10).

CONCLUSION

The results show that there are indeed differences in the results from the C++ analyzer and ESPACE. The differences are not very large, but they are significant, since the resolution of the detectors is on the order of 0.1 mm. The reason for the differences is purely due to differences in the algorithms, since both act on the same set of data. Hence, it is vital to track down the causes of the differences and determine which algorithm is, in fact, more accurate. Since the ESPACE code has been in use for many years, it is strongly expected to be more accurate at present. The most effective way to verify this would be to run both algorithms on simulated data, where the correct answer is known, rather than on experimental data.

One difference between the C++ analyzer algorithm and the ESPACE algorithm is that the angles used for the t-x conversion are computed differently. ESPACE finds a cluster and then uses characteristics of that cluster in order to compute the angle. The C++ analyzer uses the angle between pairs of clusters in the lower and upper planes, which is obtained during the coarse tracking phase. This means that the C++ results should be more reliable.

A second difference is that clusters are matched in a different order. When handling multiple track events, ESPACE first pairs

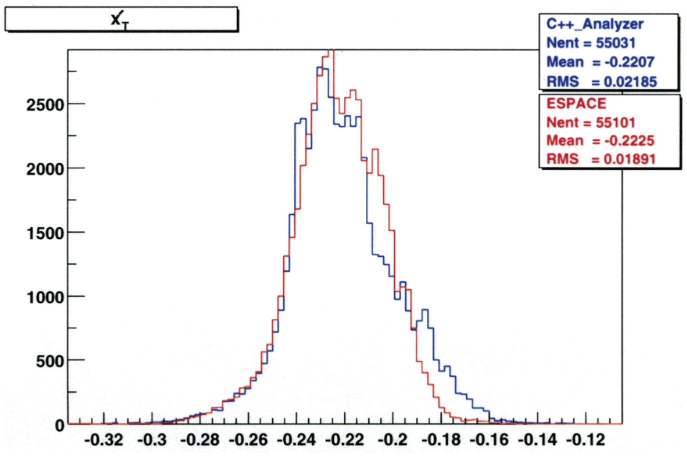

Figure 7. X'_T spectrum from a 9-foil carbon target. The blue results are from the C++ analyzer and the red results are from ESPACE. The x-axis is in meters.

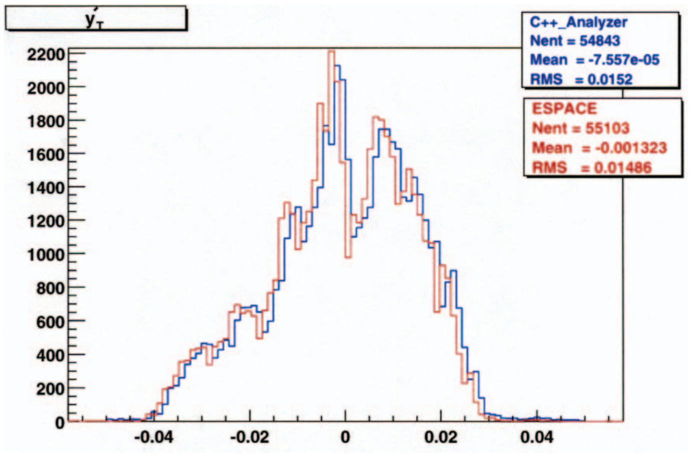

Figure 8: Y'_T spectrum from a 9-foil carbon target. The blue results are from the C++ analyzer and the red results are from ESPACE. The x-axis is in meters.

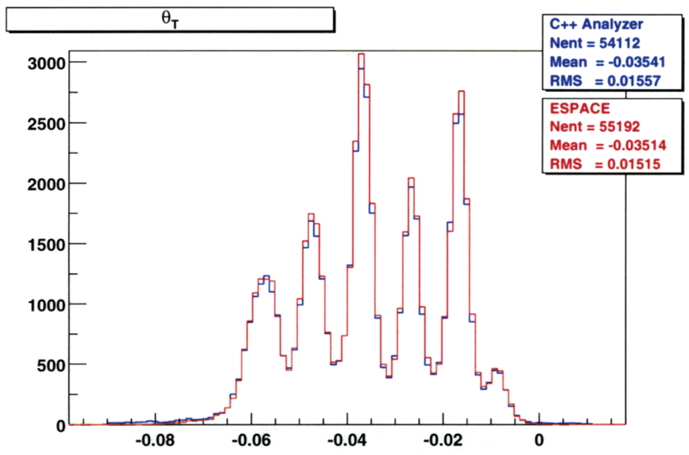

Figure 9. θ_T spectrum from a 9-foil carbon target. The blue results are from the C++ analyzer and the red results are from ESPACE. The x-axis is tan(θ_T).

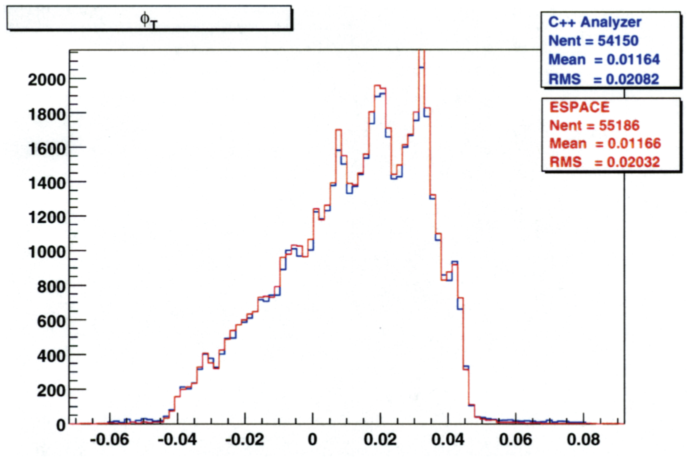

Figure 10. ϕ_T spectrum from a 9-foil carbon target. The blue results are from the C++ analyzer and the red results are from ESPACE. The x-axis is tan(ϕ_T).

clusters from the U_1 and U_2 planes, then pairs clusters from the V_1 and V_2 planes, before finally matching sets of 4 clusters into tracks. The C++ analyzer, however, matches U_1 and V_1 planes, and then matches U_2 and V_2 planes, creating lower and upper UV tracks. The lower and upper UV tracks are then paired in order to form complete tracks. The main difference is that ESPACE is able to form a more precise fit of drift distances to wire positions, since it has access to more points. Although this difference is negligible for low-data-rate experiments where less than 1% of events involve multiple tracks, it could have an impact on high-data-rate experiments.

Unfortunately, neither ESPACE nor the C++ analyzer is currently able to reliably process events with multiple particle tracks. ESPACE's current methods for handling multiple particle tracks are known to be inefficient, but work is under way to improve them. The technique used by the C++ analyzer has not yet been thoroughly tested, so it is impossible to say how efficient it is. The main advantage of the C++ analyzer is that it is being designed in a highly modular fashion, unlike ESPACE, which has many complicated interdependencies between program units. As a result, interchanging algorithms and making changes to the code when the detector configuration changes can be done more rapidly with the new C++ analyzer.

At present, there is still a great deal of work that must be done to compare the performance of the two algorithms. It is important to determine which one is actually more accurate, and to track down the differences in the algorithms leading to the current differences. The speed of the algorithms has not been rigorously tested, though they are roughly equal. A thorough test of speed should occur after the differences in the results have been accounted for and the C++ analyzer has been optimized. Since the ultimate goal of the C++ analyzer is to replace ESPACE, more testing is currently underway.

ACKNOWLEDGEMENTS

This project was performed at the Thomas Jefferson National Accelerator Facility in Newport News, Virginia. I would like to thank the United States Department of Energy, Office of Science for creating, organizing, and funding the project and the National Science Foundation for additional help with organization and sponsorship.

I would also like to thank the people who helped with this project. First, I owe a tremendous amount to Ole for tirelessly reviewing my code, helping me to understand the ESPACE code, and giving me very helpful insights on occasions too numerous to count. I should also thank Jan for making sure that I had fun during the summer. Finally, I would like to thank Bodo, Nilanga, Doug, John, Karen, and Joe for their contributions.

REFERENCES

Brown, K. L. et al. (1977). TRANSPORT: A Computer Program for Designing Charged Particle Beam Transport Systems. SLAC-91, UC-28.

Brun, R. & Rademakers, F. (2001). The ROOT System Home Page [WWW Page]. URL http://root.cern.ch

CERN. (2001). CERN Program Library [WWW Page]. URL http://cernlib.web.cern.ch/cernlib

Fissum, K.G., et al. (2001). Vertical Drift Chambers for the Hall A High Resolution Spectrometers at Jefferson Lab. Nucl. Instr. Meth. A474, 108.

Leathers, C. (1996). Efficiency Measurements on the CEBAF Hall A VDCs. Undergraduate Thesis, Massachusetts Institute of Technology. Available http://www.jlab.org/~fissum/vdcs/documentation/docs.html

Liyanage, N. K. B. (1999). A Study of the ^{16}O(e,e'p) Reaction at Deep Missing Energies. Doctoral Thesis, Massachusetts Institute of Technology. (unpublished)

Offerman, E. (1997). ESPACE User's Guide. Available http://hallaweb.jlab.org/espace/docs.html

Wechsler, R.H. (1996). Drift-Time Properties of CEBAF Hall A Vertical Drift Chambers. Undergraduate Thesis, Massachusetts Institute of Technology. Available http://www.jlab.org/~fissum/vdcs/documentation/docs.html