DIMAD Based Interactive Simulation of the CEBAF Accelerator*

M. H. Bickley and D. R. Douglas
Continuous Electron Beam Accelerator Facility
12000 Jefferson Avenue
Newport News, VA. 23606
R. V. Servranckx
TRIUMF
4004 Wesbrook Mall
Vancouver, B.C. V6T243

ABSTRACT

An X-Windows^{TM} based interactive interface to the DIMAD beam optics program[1] enables users to simulate the adjustment of magnets in tuning various segments of the CEBAF beamline. In addition, users can track the effects of random errors on the path of individual particles as magnets are adjusted. The interface sits on top of the standard DIMAD model, retaining the detailed modeling available with that code. Because X-Windows software was used, the code is portable to any system that has X-Windows and the X-Windows Toolkit available. We give results from the studies simulating the extraction portion of the CEBAF beamline.

INTRODUCTION

The computer program DIMAD is a widely used tool for the analysis and design of particle beam transport systems. The standard model possesses an interface that allows users to interactively vary the values of hardware parameters and view the resulting measurements taken at selected beam monitors in the beamline being simulated. Users enter data by typing characters at a keyboard. The characters are used to select options from a menu, to add incremental values to hardware parameters or to change the size of an increment. The output is alphanumeric, displaying a histogram indicating the measured beam values at the monitors. The existing alphanumeric interface is limited in several respects: The entering of data is not intuitive. The display has low resolution. The interface is hardware-dependent, requiring recoding whenever a new kind of terminal is used.

A new graphical interface was needed to remedy these shortcomings. The software to accomplish this has been implemented and is discussed in the following.

*Supported by D.O.E. contract #DE-AC05-84ER40150

DESIGN AND IMPLEMENTATION

Design Goals

The new interface was designed with five goals in mind:

1) Retain all functions of the old DIMAD interface. This assures that all required features will be available and users of the previous interface will have little difficulty with the new one.
2) Make the interface hardware-independent. This is done to simplify software portability and installation efforts. It allows usage on a wide variety of machines. This goal requires a standard software base, so C and X-windows were selected for their near-universal availability on workstations.
3) Provide an interface that is as easy to use and as intuitive as possible. Try not to burden the user with insignificant details about using it.
4) Limit the changes to DIMAD. Given the fact that there are many different versions of DIMAD, try to limit the work required to make any version compatible with the new interface. This also keeps the interaction between DIMAD and the interface small and well-defined.
5) Use modular coding techniques. Simplify any future work associated with the extension or alteration of the interface.
   The entire process, from design to implementation through testing and debugging, took approximately three man-months.

Changes to Existing DIMAD Code

There were three phases of implementation of the new interface with respect to the standard DIMAD code. First, the raw information to be displayed was gathered and organized. Second, the flow of control was directed to the new code to display the graphical interface. Third, user input from the interface was interpreted and the appropriate functions executed by DIMAD. Each of these steps, outlined below, required very little modification to existing code.

Accomplishing the first phase required the construction of a data structure to hold the DIMAD data needed by the interface. The raw data was collected by identifying code locations in which the data had already been calculated or else was available from the processing of other data.

The second phase was simple to implement. Since an interactive interface was already in existence, the new interface required only that it be initialized and that the flow of control be directed to the new code.

The third phase was the most difficult with respect to designing the interface. The new software requires the reconciliation of two disparate coding techniques–FORTRAN's linear programming and X-Windows' event-driven structure. This conflict was resolved by taking advantage of the availability of FORTRAN recursion on workstation-class machines. This enabled all DIMAD functions executed by the new interface to be retained in the same single subroutine in which they were implemented for the old alphanumeric interface. The alternative would have been to break the single routine into many pieces, with each piece responsible for one type of user input from the new interface.

New Code Using X-Windows

Building the graphical interface required the use of all of the levels of abstraction available with X-Windows. The code was implemented by utilizing the highest level possible for each segment, minimizing the programming work. This means that Motif, the highest level, was used to implement widgets such as buttons, toggles, scales and labels. These standard widgets were extended when necessary. For instance the Motif scale was not flexible enough for the purpose of setting hardware parameters, so several Motif buttons and labels were added to make a more "intelligent" widget.

At a lower level of abstraction, the X-Intrinsics were used for managing the various windows and widgets as much as possible. This served to minimize programming work by relying on the particular implementation of X-Windows to handle these objects.

At the lowest level, managing the graphical windows, X-Library functions were used. The lines and characters in these windows are each explicitly drawn by the interface software, so they must be handled at the most basic level.

The implementation and testing of the new X-Windows code consumed the majority of the time needed to create the new interface.

TESTING THE SOFTWARE

The interface software was tested by simulating a very simple beamline which consisted of six ten-meter drifts alternated with simple dipole correctors and beam monitors. The "beam" was made up of twenty test rays. Initially, the beam centroid at the monitors was very close to zero. The non-zero values are a function of the statistical population of the beam with energies and positions. The beam width at the monitors increased linearly. As small changes were made to the strengths of the correctors the beam centroid as indicated by the graphical display would bend an amount equal to that calculated independently. Figure 1 shows the horizontal beam centroid values measured at monitors in the beamline. In this example a horizontal corrector immediately after the second monitor has been given a small positive value.

The angle of deflection was measured, and it was identical to the expected value given the strength of the correction.

Further tests using the simple case showed the deflection from a corrector could be exactly compensated for by giving a corrector further down the beamline a strength equal to and opposite in sign from that of the first corrector, resulting in a beam centroid that traveled parallel to the beam centerline. The effects of the correctors contributed additively, as expected. Also as expected, the beam width was not affected by changes to the strengths of the correctors–no focussing effects were being modeled.

The same basic beamline was used for testing the measurement of focussing. By adding a quadrupole to the simple beamline it was demonstrated that focussing effects of magnetic elements were accurately reflected by changes in the beam width.

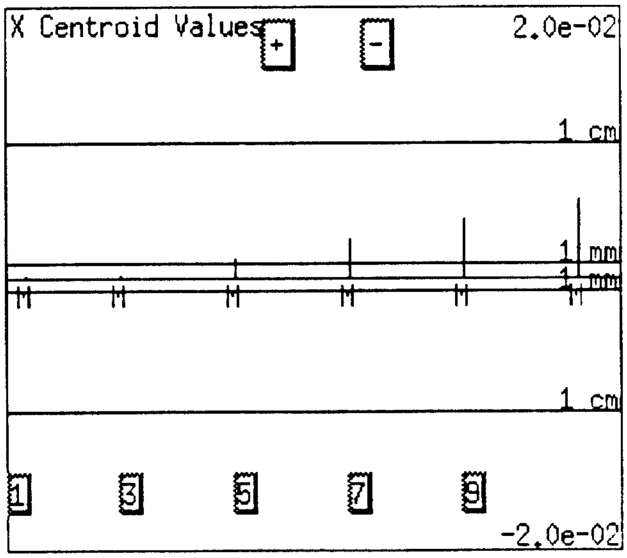

Figure 1. Test of the interface software

EXAMPLE PROBLEM

The new software was used in the analysis of the extraction region of the CEBAF accelerator. The purpose of the study was to determine the strengths required for two septa in order to attain an extracted beam offset of 20 centimeters, and to verify that the clearances of the beam and the septa were large enough. The beamline being simulated included the 80 meters of line upstream of the thin septum, and the 35 meters downstream of the thin septum.

The analysis was performed by giving each magnet in the line random misalignments, where the magnitude of the misalignments was based on a gaussian distribution whose sigma was equal to the sigma predicted for the real CEBAF magnets after installation. The simulation included an RF separator element, which took each test ray and created two new ones with identical characteristics. One of the two rays was given a positive horizontal kick and the other was given a negative horizontal kick, generating two beams downstream of the RF separator where there was only one beam prior to it.

The above procedure was followed repeatedly ten times (with a different starting random number seed for each), and the results displayed on the graphics screen. Figure 2 is a hard copy of the results of the simulation. The thin septum is represented as a thick vertical line with a gap through which the upper portion of the beam passes. The magnification of the display is such that only the wall of the thin septum is shown. The resolution is high enough that each test ray trajectory is discernible.

Figure 3 shows the same length of simulated beamline, however the magnification has been changed so that the beam can be seen clearly passing through the thick septum. Here the resolution does not permit distinguishing separate trajectories, however the envelope of the paths shows that the beamline safely passes both septa.

CONCLUSIONS

The substitution of the new interface for the old was a simple process in this case. If the old interface had been spread out among a number of routines the implementation procedure would have been more complicated, but even so would have been straightforward in execution.

In general, the benefits of a graphical interface become less and less important as the response time of the software increases and users must wait too long for the results of iterative changes. The response time is quite good for DIMAD used with this interface. Our typical analyses use 100 to 500 beam elements in the line and 20 to 100 test rays in the beam. The response time on a DECstation 3100 is well under a second in the best cases to about ten seconds in the worst. At its worst, the response is still quite adequate and allows a painless level of interactivity with the model DIMAD.

ACKNOWLEDGMENTS

We would like to thank Dr. Richard York for initiating the development of this software.

REFERENCES

[1] R. V. Servranckx, *et al.*, "User's Guide to the Program DIMAD," SLAC Report 285 UC-28, May 1985.

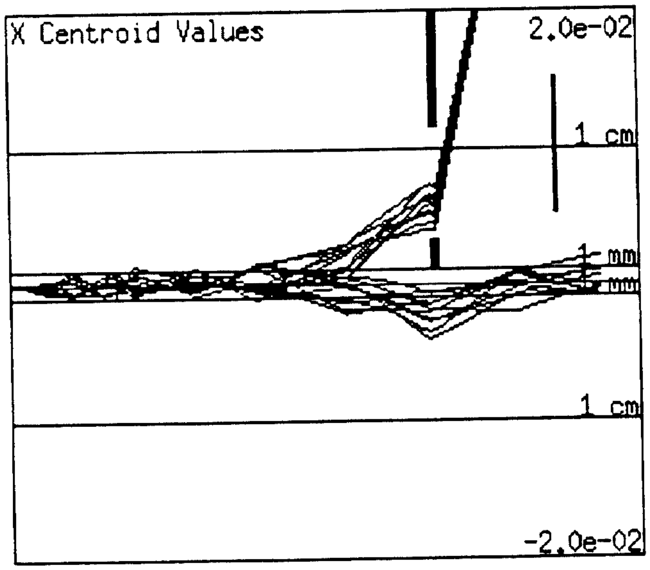

Figure 2. Close-up view of the extraction region

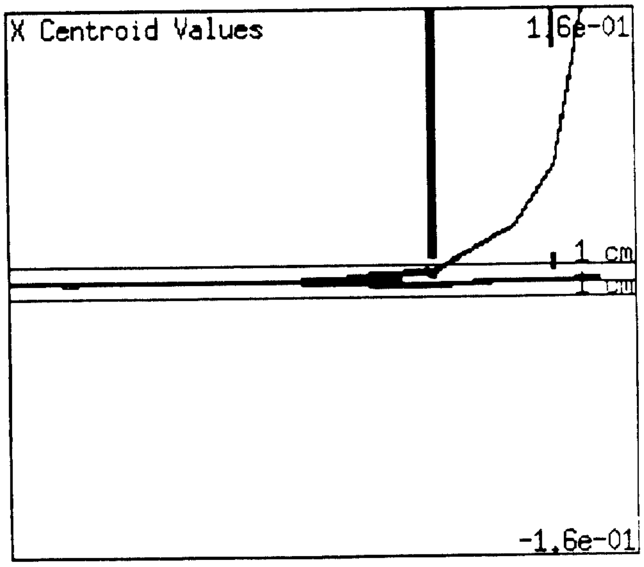

Figure 3. Beam passing cleanly through septa in the extraction region