

To the Graduate Council:

I am submitting herewith a thesis written by Reuben Dale Hochstedler entitled "Acceleration of the Integrated TIGER Series of Coupled Electron/Photon Monte Carlo Transport Codes System." I have examined the final copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Electrical Engineering.


Dr. L. Montgomery Smith, Major Professor

We have read this thesis
and recommend its acceptance:





Accepted for the Council:


Associate Vice Chancellor
and Dean of The Graduate School

ACCELERATION OF THE INTEGRATED TIGER SERIES OF COUPLED ELECTRON/PHOTON MONTE CARLO TRANSPORT CODES SYSTEM

A Thesis
Presented for the
Master of Science
Degree
The University of Tennessee, Knoxville

Reuben Dale Hochstedler
December 1996

DEDICATION

This thesis is dedicated to my parents

Robert Ivan Hochstedler

and

Rachael Marie Hochstedler

who never limited me
with any preconceived notions
of what I could or could not achieve.

ACKNOWLEDGMENTS

I would like to acknowledge the contributions, advice and diligence of my major professor, Dr. L. Montgomery Smith. His assistance and patience were instrumental to the completion of this thesis. I would also like to thank my committee members Dr. Bruce Bomar and Dr. Bruce Whitehead for their helpful suggestions and comments in refining this work. Extra thanks goes to Dr. Whitehead for helping to secure financial funding for my final semester at UTSI. I want to thank my parents Robert and Rachael Hochstedler for helping me to get this far; I could never have made it without them. And finally, I wish to thank Tamara Douglas for all of her encouragement, support and belief in me.

This work was supported by the United States Air Force, Arnold Engineering Development Center under Contract No. F40600-94-D-0001, Tasks 95-02 and 96-02, Arnold AFB, TN 37389.

ABSTRACT

The Integrated TIGER Series (ITS) of coupled electron/photon Monte Carlo transport codes system has been accelerated. This was accomplished by examining the program for inefficient coding and replacing the inefficient methods with more computationally efficient ones.

Each member code of ITS was benchmarked and profiled with a specific test case. The profiling results directed the acceleration efforts towards the most computationally intensive subroutines. Techniques for accelerating these subroutines were identified and included replacing linear search algorithms with binary versions, replacing the pseudo-random number generator, reducing program memory allocation, and proofing the input files for geometrical redundancies. The accelerated code produced either identical or statistically similar results to that of the original code system.

Final benchmark timing of the improved ITS showed a speed up factor of 2.00 for TIGER (the 1-D slab geometry code), 1.74 for CYLTRAN (the 2-D cylindrical geometry code), and 1.90 for ACCEPT (the arbitrary 3-D geometry code). An extra step was added to the use of ITS, but this extra step also increased the usefulness of ITS as a design tool. These improvements are machine portable and easily integrated into an existing ITS code installation.

TABLE OF CONTENTS

<u>Chapter</u>	<u>Page</u>
I. INTRODUCTION.....	1
II. BACKGROUND.....	4
A. <i>The DECADE Pulsed Power Facility</i>	4
B. <i>Integrated TIGER Series, version 3.0</i>	5
III. METHODOLOGY FOR IMPROVING THE COMPUTATIONAL EFFICIENCY.....	7
A. <i>Benchmarking the Original ITS Code</i>	9
B. <i>Profiling the Original ITS Code</i>	10
C. <i>Identifying and Improving Computational Efficiency</i>	10
1) <i>The Quadratic Interpolation Routine</i>	12
2) <i>The Random Number Generator</i>	14
3) <i>Memory Requirements</i>	17
4) <i>3-D Geometry</i>	18
D. <i>Benchmarking the Improved Code</i>	20
IV. RESULTS.....	21
A. <i>Additional Benefits</i>	23
B. <i>Negative Effects</i>	24
V. CONCLUSIONS AND FUTURE WORK.....	25
REFERENCES.....	28
APPENDICES.....	30
A. <i>ITS Test Case Input Files</i>	31
1) <i>TIGER Input Files</i>	32
2) <i>CYLTRAN Input Files</i>	33
3) <i>ACCEPT Input Files</i>	35
B. <i>UPEML Update Directives</i>	37
C. <i>Improved ACCEPT Test Case Input File</i>	52
D. <i>ACCEPT Monte Carlo Geometry Efficiency Verification</i>	55
E. <i>ACCEPT Analytical Geometry Efficiency Verification</i>	57
VITA.....	75

I. INTRODUCTION

THE Integrated TIGER Series (ITS) of coupled electron/photon Monte Carlo transport codes is a powerful tool for determining state-of-the-art descriptions of the production and transport of the electron/photon cascade in time-independent, multi-material, multi-dimensional environments [1-3]. ITS (and codes like it) are established methods for analysis in such applications as radiation shielding calculations, x-ray environment modeling, and others where it is necessary to compute dose rates of ionizing radiation in matter.

ITS is a collection of programs sharing a common source code library that can solve sophisticated radiation transport problems using a particle-by-particle, random-number-based, statistical approach (Monte Carlo method). A total of eight codes are in the collection which can be split into six groups: the TIGER codes (for 1-D slab geometries), the CYLTRAN codes (for 2-D cylindrical geometries), the ACCEPT codes (for arbitrary 3-D geometries), the standard codes (for normal applications), the P codes (for applications where enhanced ionization/relaxation procedures are needed), and the M codes (for applications which involve 2- or 3-D macroscopic electromagnetic fields). The user selects the appropriate code from the library and supplies it with any special requirements and the physical description of the problem to be solved in an input file. Upon completion of the calculations, the results are stored in an output file.

However, individual problems posed to ITS can take many hours to complete. For instance, the test problem that was used for TIGER in this study took nearly seven hours of computation time to complete. This is due to the number of particle histories required for accuracy, the computational complexities involved in the Monte Carlo approach (such as random number generation), and any inefficient programming present in the codes. As a result, while ITS is a good verification tool, it is not an effective design tool for radiation environment modeling using a trial-and-error approach.

The particular application that gave rise to the work presented here is the prediction of the test cell environment of the DECADE pulsed-power nuclear weapons effect simulator being constructed at the Arnold Engineering Development Center (AEDC) [4].

ITS was selected for this task and has previously been used to compute the output of the front-end diode for comparison with experimental findings [5]. In the future, it will be used for analysis and design of production-line tests in designated DECADE facility test chambers [6].

ITS 1.0 was initially released in December 1984. Before the initial series release, the individual member codes were separate entities. The major thrust of the first release was to merge these codes into one common source library. This made the codes easier to use and maintain, and implemented in all the codes the advanced features which were spread out among them. Since version 1.0 of ITS, "subsequent versions have implemented important improvements in physical accuracy, new capabilities, variance reduction, and user friendliness" [1]. There is no indication that efforts have been taken specifically to increase the efficiency of ITS in any of the subsequent versions up to and including the current 3.0 version.

The objective of the work presented here was to reduce the required execution times of ITS when solving a radiation transport problem. This was accomplished by examining the ITS source code for computational inefficiencies. Once inefficiencies were identified, they were reprogrammed to increase their execution speed while preserving their original function and output. Finally, the modified source code was tested to ensure that it produced the same results as the original. None of the ITS radiation modeling techniques were modified, only its computational efficiency was improved.

ITS was successfully accelerated through the efforts described here. A speed up factor of 2.00 was achieved for the TIGER test case, 1.74 for the CYLTRAN test case, and 1.90 for the ACCEPT test case. All techniques applied were written in FORTRAN 77; therefore, the modified version is as portable as the original. Furthermore, none of the functionality of ITS was removed to achieve this speed up.

The acceleration techniques used included replacing linear searches with binary searches, a new pseudo-random number generator, memory reduction, and proofing the input files for ACCEPT for geometrical redundancies. The memory reduction and proofing resulted in an added step in the use of the ITS system. However, these techniques also

produced some side benefits not directly related to computation time that improved the resource allocation and user-friendliness of ITS and its usefulness as an engineering tool.

Chapter 2 will provide background information on the DECADE project at AEDC and an overview of how the Integrated TIGER Series is used. Chapter 3 describes how ITS was analyzed to find programming inefficiencies, what inefficiencies were identified, and what methods were used to speed up the execution time of the codes. Chapter 4 discusses the results of the work from Chapter 3. Negative effects and non-speed benefits are discussed in addition to the speed up achieved. Chapter 5 makes conclusions and describes possible future work that could be applied to further speed up ITS.

Appendices have been included to allow duplication of the work presented here. Appendix A contains the test case input files used for the TIGER, CYLTRAN and ACCEPT codes. Appendix B is the majority of the programming changes used to achieve execution speed up. Appendix C has the improved ACCEPT test case input file which was a result of the speed up work. The computer codes for verifying the efficiency of the 3-D geometry specifications in ACCEPT input files are stored in the remaining two appendices. The Monte Carlo version is in Appendix D; the deterministic version is in Appendix E.

II. BACKGROUND

In this chapter, background information on the DECADE pulsed power facility is given. This describes the motivation for this study. Background information on the Integrated TIGER Series is given to describe the basics of how to use ITS.

A. *The DECADE Pulsed Power Facility*

DECADE is a nuclear weapons effect simulator, the largest such simulator in existence. It is based around a "35 TW pulsed power generator designed to drive both bremsstrahlung and plasma radiation source loads" [4]. It is used to simulate the radiation effects in the materials placed in the test chamber that a nuclear weapon would produce. This is important in designing and testing technologies that are to survive "nearby" nuclear explosions, especially in space.

The DECADE generator consists of sixteen independent pulsed power modules. Each module consists of a 500 kJ Marx that pulse charges a water transfer capacitor. The transfer capacitor discharges into a water output line through an array of triggered gas switches. The water output line then pulse charges the final inductive store/opening switch pulse compression stage. When the opening switch opens, the inductive store discharges into either an independent diode load or into a vacuum convolute that connects all sixteen modules to single imploding plasma load....

In the large-area configuration, the average dose will be 20 kRad (in silicon), delivered over an area of 1 square meter. The maximum endpoint voltage for this case will be 1.5 MV, and the radiation pulse-width (FWMH) will be 40 ns. The high-dose configuration will deliver 80 kRad into 1500 cm². For this case, the end-point voltage is 1.8 MV; the radiation pulse width is also 40 ns.... DECADE will deliver 2.5 MJ to the load in 140 ns, with a peak voltage of 1.8 MV, a diode current of 22 MA, and a diode power of 35 TW. [4]

To be a world-class, high-productivity simulation facility, DECADE must provide not only extensive experimental support but also theoretical simulation support. For theoretical simulation of the radiation environment inside the test chamber, version 3.0 of the Integrated TIGER Series was chosen. ITS had already been used to verify experimental

tests of the front-end diode used to produce bremsstrahlung radiation in the DECADE simulator [5] as well as many other radiation applications [7-8].

The theoretical radiation environment simulation would also be used for analysis and design of production-line tests before actual tests in the chamber. It was therefore desired to use ITS to observe the effects of design changes in the final simulation. This desire to use ITS as a trial-and-error design tool meant improvements in its execution speed would be essential.

B. Integrated TIGER Series, version 3.0

ITS is a powerful and user-friendly software package permitting state-of-the-art Monte Carlo solution of linear time-independent coupled electron/photon radiation transport problems, with or without the presence of macroscopic electric and magnetic fields. It combines operational simplicity and physical accuracy in order to provide experimentalists and theorists alike with a method for the routine but rigorous solution of sophisticated radiation transport problems. Flexibility of construction permits tailoring of the codes to specific applications and extension of code capabilities to more complex applications through simple update procedures. [2]

Only the basics of understanding and using ITS will be discussed here. For more detailed information on ITS, consult [1].

ITS is a collection of eight different computer codes that all share a common source code library. The eight codes can be separated into the three groupings in two different ways. The first separation is by the geometry of the environment being modeled; the one-dimensional (slab geometry) codes are called the TIGER codes, the two-dimensional (cylindrically symmetric geometry) codes are called the CYLTRAN codes, and the three-dimensional (arbitrary shape geometry) codes are referred to as the ACCEPT codes. The second separation is by function; there are the standard codes, the P codes (which include code for enhanced ionization/relaxation), and the M codes (which include code for macroscopic electromagnetic fields [There is no one-dimensional version.]). From this point forward only the first separation method will be referred to.

The different member codes are extracted from the source code library through the use of a machine-portable CDC update emulator called UPEML which is included in the ITS distribution. UPEML accomplishes this extraction through a technique similar to that of the C language preprocessor. The user defines in an input file which of the eight FORTRAN source codes is to be extracted from the source library and which of four computing environments the code is being targeted (CDC, CRAY, IBM or VAX). Then UPEML searches the source library for update directives (analogous to C # directives) which are inserted in-line in the library. These directives tell UPEML which parts of the library are appropriate for the desired code and machine type. UPEML then places the FORTRAN source code in an output file which the user can then compile with a FORTRAN 77 compiler.

UPEML also has the ability to modify the output source code through additional update directives supplied by the user at code extraction time. This is how ITS maintains its extensibility and customization features. This allows for users to change the final compilation file while still having easy access to the original ITS source code if necessary.

Besides the eight simulation codes already described, ITS also includes a preprocessing program XGEN which generates the material cross-sections used by the simulation codes. The user gives XGEN the elements and compounds present in the problem in an input file and XGEN places the cross-sectional data in specific output files to be read by the simulation code. Acceleration of this program was not addressed. The output of XGEN can be used in multiple simulations that use the same materials, and since its execution typically takes only a few minutes, it was not an important issue in the work presented here.

After the appropriate simulation code has been generated and compiled, and once the material cross-section has been produced, the user places the physical description of the problem to be solved along with any solution constraints into an input file. The simulation code is then executed. Upon completion, the final results including statistical analysis are placed in a human-readable output file.

III. METHODOLOGY FOR IMPROVING THE COMPUTATIONAL EFFICIENCY

ITS is a very large and complex computer code system. It is not a simple subroutine which can be visually inspected for inefficient coding. A much more analytical and rigorous approach must be taken. The following approach was taken to speed up ITS.

Realistic test cases were defined, one each for the three different member codes of ITS. For TIGER, the one-dimensional code, the test case was a 1 MeV electron flux incident on a tantalum/aluminum x-ray diode in front of an aluminum-shielded silicon detector as illustrated in Fig. 1. For CYLTRAN, the two-dimensional code, the test case was taken from Appendix A of [5]. It was a 0.3 MeV electron flux incident on a tantalum/aluminum diode like the TIGER case but put into a cylindrical test cell with a circular lithium fluoride thermoluminescent detector array at the opposite end as illustrated in Fig. 2. For ACCEPT, the three-dimensional code, the test case was supplied by the Air Force. It was a photon flux incident on a concrete wall in air with an off-center stainless-steel/lead-lined hole as illustrated in Fig. 3 [6]. The input files for each of the test cases are included in Appendix A.

These test cases were benchmarked to determine baseline times and to obtain output files for later comparison with modified versions of the code. Next, the three different codes were profiled to determine which subroutines should be targeted for improvement. Once identified, each targeted subroutine was investigated to find any computational inefficiencies that may have existed. If an inefficiency was found, a more computationally efficient algorithm was developed to replace it. The replacements were tested to verify their correctness against the originals. Once all of the new algorithms were in place, the modified codes were benchmarked against their original versions to determine the overall speed up achieved.

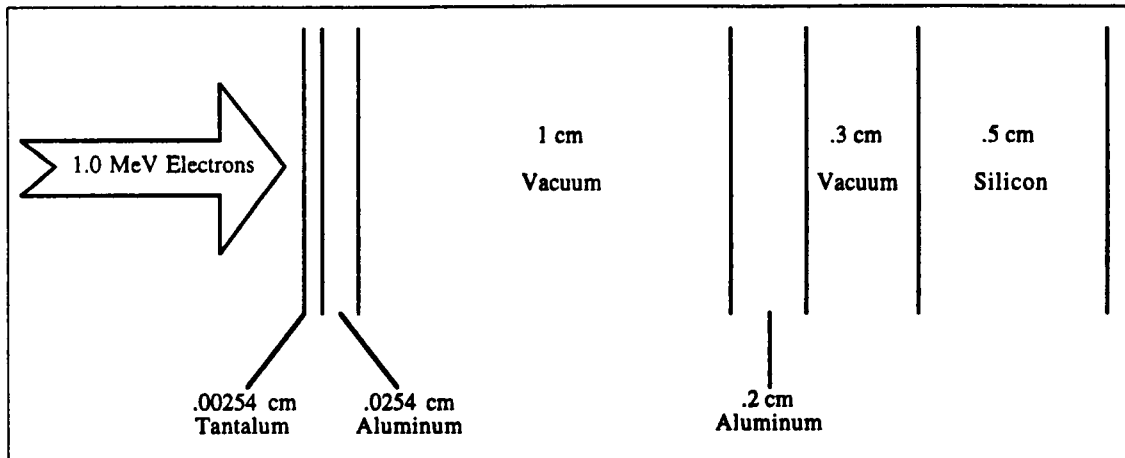


Fig. 1. TIGER (1-D) Test Case Geometry. Ta/Al x-ray diode incident on an Al-shielded Si detector. Horizontal axis relates to distance from source. Image is not to scale.

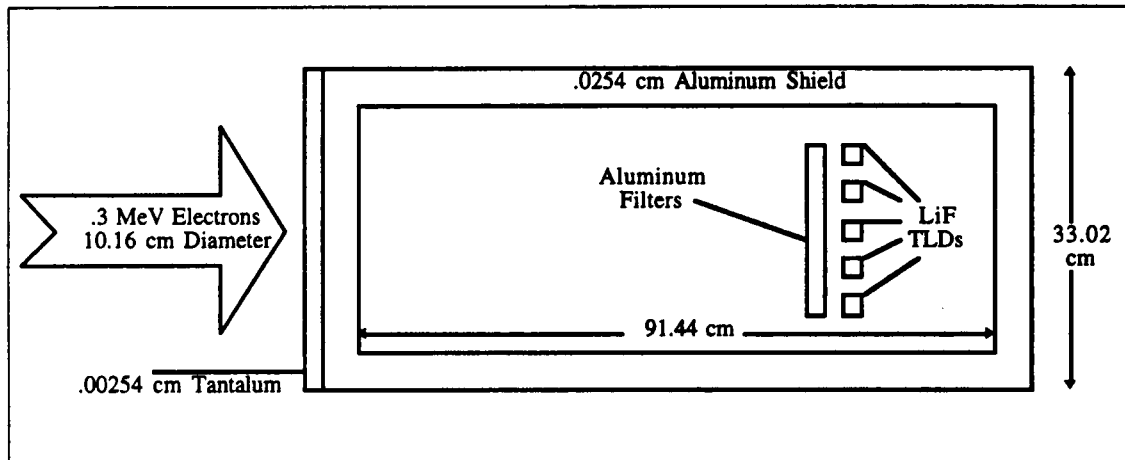


Fig. 2. CYLTRAN (2-D) Test Case Geometry. Ta/Al x-ray diode input to a cylindrical test cell with a circular array of LiF TLDs. Horizontal axis relates to distance from source. Vertical axis indicates axial symmetry. Image is not to scale.

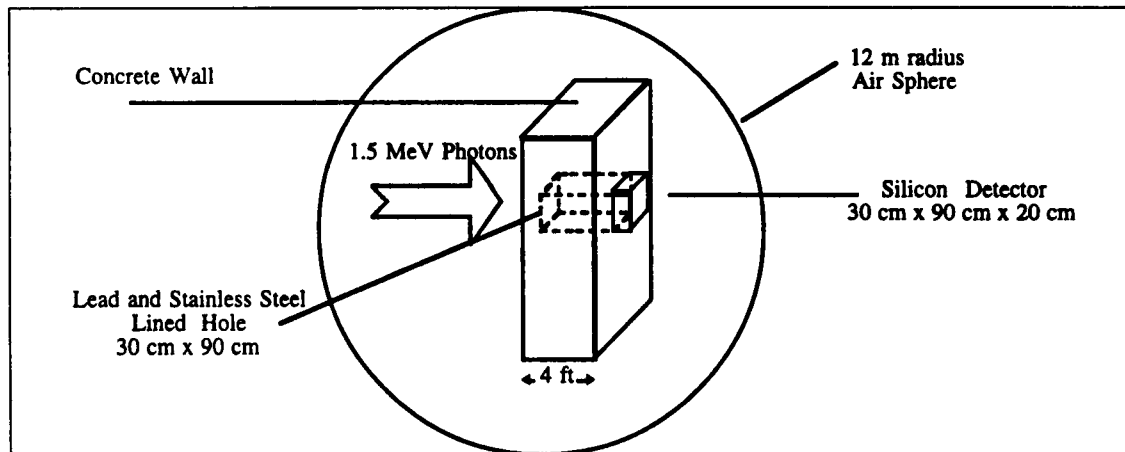


Fig. 3. ACCEPT (3-D) Test Case Geometry. Photon flux incident on a concrete wall in air with a stainless-steel/lead-lined off-center hole with silicon detector. Image is not to scale and for visualization purposes only.

A. Benchmarking the Original ITS Code

The three different member codes of ITS (TIGER, CYLTRAN and ACCEPT) were timed using 800,000 particle history versions of each of their individual test cases. This benchmarking was performed on a Sun SPARCstation 10 running under Solaris 2.4 (SunOS 5.4). The codes were compiled using version 2.0.1 of the Sun f77 FORTRAN compiler with full optimization (-O4 -dalign -native). The standard LINPACK rating for this machine is 10 Mflop/s (compiled with -fast -O4 -unroll=4 -Bstatic). The "Toward Peak Performance" LINPACK rating is 27 Mflop/s [9].

The internal timing mechanism of ITS was used so that only the radiation transport simulation was taken into account and not any of the program's input/output routines which would have added an unimportant constant amount to the overall timing. Oddly, the overall time reporting mechanism of ITS reported 0.0 seconds every time. Since the overall timer was non-functional, the reported average time per batch was multiplied by the number of batches (in all cases 10) to determine the overall time used. These times are as given in Table I.

TABLE I
BENCHMARK TIMES OF ORIGINAL ITS MEMBER CODES
ALL TIMES IN HOURS:MINUTES:SECONDS

TIGER	CYLTRAN	ACCEPT
6:45:31.41	5:24:04.91	4:23:45.42

The output files from these benchmarking runs were saved for later comparison with the modified versions. The comparison is to ensure that the modifications give identical results to the original code, or at least statistically similar to each other when identical results could not be expected.

B. Profiling the Original ITS Code

Once baseline times and output were determined for each of the test cases, the next step was to determine which of the subroutines in each of the ITS codes should be examined for inefficient coding.

Profiling was accomplished by compiling each of the ITS codes with the built-in profiling mechanism of the f77 FORTRAN compiler (-p command line option). For the profiling runs 1000 particle history versions of each of the test cases were used. The profiling output was then transformed into readable form and examined.

The reported profile included timing data for many of the intrinsic functions of FORTRAN as well as the program's subroutines. These intrinsic functions cannot be modified except at the assembly code level. Since this would result in non-portability to other computer platforms and since the functions were most likely efficient as written, they were removed from consideration.

Since the profiling run was only for 1000 particle histories, the time taken in input/output routines was comparable to that taken in the simulation routines. In normal use of ITS the amount of time spent in input/output routines is negligible. These routines were also removed from consideration.

This left the subroutines of interest, the largest time consumers being at the top of the list. The top time consuming subroutines of interest for each ITS code and their basic function are listed in Table II.

C. Identifying and Improving Computational Efficiency

All of the subroutines listed were examined for inefficiencies. Most of the subroutines could be grouped into four basic categories: administrative routines, those that depended on searching a tabulated array of data and then interpolating a result, the pseudo-random number generator, and those that were particular to ACCEPT's three-dimensional nature. The administrative routines were already efficiently coded and not modified.

TABLE II
 TOP TIME CONSUMING SUBROUTINES FOR THE ITS CODES
 FIRST LETTER INDICATES CATEGORY, A=ADMINISTRATIVE, S=SEARCH AND INTERPOLATE
 R=RANDOM NUMBER GENERATION AND 3=3-D GEOMETRY.
 SECOND LETTER INDICATES AMOUNT OF MODIFICATION, H=HEAVY, L=LIGHT, N=NONE

TIGER	QPOL	SH	quadratic interpolation
	PHIST	AN	photon history tracking
	RAN	RH	pseudo-random number generator
	BANGLE	SH	scattering angle
	EHIST	AN	electron history tracking
	BREMS	N	brehmsstrahlung production
	PLACE	N	geometry
	MULT	SH	multiple scatter
	EQDEP	SH	equilibrium

CYLTRAN	QPOL	SH	quadratic interpolation
	EHIST	AN	electron history tracking
	RAN	RH	pseudo-random number generator
	PHIST	AN	photon history tracking
	BREMS	N	brehmsstrahlung production
	MULT	SH	multiple scatter
	BANGLE	SH	scattering angle
	FOLD	L	scattering direction in lab frame
	ZONE	SH	sub-zone geometry location

ACCEPT	GG	3 L	3-D geometry
	EHIST	AN	electron history tracking
	DISTA	3 N	distance to interaction calculation
	QPOL	SH	quadratic interpolation
	MULT	SH	multiple scatter
	FOLD	L	scattering direction in lab frame
	LOSS	N	particle absorption
	FOLNOK	L	scattering angle calculation
	BREMS	N	brehmsstrahlung production

The most time consuming of the tabulated data routines was QPOL, the generic quadratic interpolation routine. All of the other tabulated data routines (BANGLE, EQDEP, MULT, ZONE and CLASS) have been grouped into the discussion on QPOL which was representative of the group and had some other improvements made to it. (CLASS was not a top time consumer for any particular code but was slow enough to be noticed and included in this grouping.)

The pseudo-random number generator is always one of the most crucial subroutines in any Monte Carlo based simulation code. While a large number of generators are readily available, care must be taken to choose a generator which works well with the particular simulator in which it is to be placed [10].

Some routines examined did not fall into one of the three categories already listed such as FOLD and FOLNOK. These were relatively straightforward routines with little room for improvement other than converting some mathematical squaring functions into multiplication and removing some redundant calculation of array indices. This was done although no appreciable performance gains were noticed.

Another possible method of acceleration was conceived separately from the profiling results. ITS must keep track of very large amounts of data during its simulation. The amount of data is determined at run time when the individual simulation case is read in by the program. Because ITS is written in FORTRAN 77, the amount of memory allocation is pre-determined by the source code. Thus, the amount of memory allocated to it is necessarily more than needed for any one particular simulation run. A way of circumventing this shortcoming was investigated.

Lastly the issues involved in working arbitrarily in all three spatial dimensions were examined. Some interesting findings concerning ITS's "user friendliness" were discovered.

1) The Quadratic Interpolation Routine: The work performed on the subroutine QPOL is described as an example of the re-coding performed to increase computational efficiency in a number of subroutines. QPOL is a generic quadratic interpolation routine. It searches a table for the three closest values to the input, and then fits a parabola through the three tabulated values to estimate the return value based on the location of the input.

There are two basic parts to the subroutine: locating the points closest to the point of interest that reside in the tabulated data and performing the actual interpolation. Methods for improving both were discovered. The improvements to the search algorithm are also applicable to the similar subroutines BANGLE, CLASS, EQDEP, MULT and ZONE. These routines used a linear instead of quadratic interpolation in their final calculation which was already coded efficiently.

Linear versus Binary Search: The type of search used in the original version of QPOL and the other subroutines like it was a linear search. The search first determined if the data in the table was non-decreasing or non-increasing. It then went through the table one entry at a time keeping track of when it passed the input value the search was based on. Regardless of where the closest values were located in the table, the search proceeded through the entire table. This search also made special allowances for finding exact matches, an extremely unlikely occurrence. This gives the time order of the search as $O(n)$ where n is the length of the tabulated array being searched.

This linear search algorithm was replaced by a non-recursively implemented binary search. This search first determines if the data in the table was non-decreasing or non-increasing (as in the linear search). It then looks at the mid-point of the table and compares it with the value it has been asked to find. Depending on the numerical relationship of the two values and the ordering of the data, the algorithm defines the mid-point as the top or bottom of the data table for its next pass. This continues until the top and bottom are adjacent entries in the table. These adjacent entries surround the value being looked for (except in the extremely rare case where there is an exact match).

This technique is called a binary search because it successively splits the table into two sub-sequences. This has the effect that no matter where in the table the search value lies, the algorithm will find it in the same amount of time. This results in a time order for the algorithm of $O(\log_2 n)$, where n is the length of the tabulated data array. This means that as larger and larger tabulated data arrays are used, the time savings over the old linear search algorithm grow exponentially.

Expanded versus Compressed Interpolation Computation: Once the surrounding values have been found, the third closest value in the table is determined. The subroutine then moves on to the actual interpolation calculation.

The original version of QPOL used a direct implementation of quadratic interpolation to compute the final return value. The equation is as follows:

$$y = \frac{(x - x_2)(x - x_3)}{(x_1 - x_2)(x_1 - x_3)} y_1 + \frac{(x - x_1)(x - x_3)}{(x_1 - x_2)(x_3 - x_2)} y_2 + \frac{(x - x_1)(x - x_2)}{(x_3 - x_1)(x_3 - x_2)} y_3 \quad (1)$$

where x is the input value; x_1 , x_2 and x_3 are the closest values in the table to x ; y_1 , y_2 and y_3 are the corresponding return values for x_1 , x_2 and x_3 ; and y is the return value for the function. This required 14 additions/subtractions, 6 multiplications and 6 divisions for a total of 26 double-precision floating-point operations.

By combining terms the same result can be obtained from the following equation.

$$y = y_2 + \left(\frac{x - x_2}{x_3 - x_1} \right) \left[\frac{(x - x_1)(y_3 - y_2)}{x_3 - x_2} + \frac{(x_3 - x)(y_2 - y_1)}{x_2 - x_1} \right] \quad (2)$$

This required 10 additions/subtractions, 3 multiplications and 3 divisions for a total of 16 double-precision floating-point operations.

This had the effect of cutting the time for the interpolation calculation nearly in half while maintaining significant equivalency between the original and the modified version of QPOL. In fact, when output files from identical runs were compared, there were no differences between the two versions except for the amount of time used.

2) *The Random Number Generator:* The pseudo-random number generator is one of the most important aspects of ITS since random numbers are the foundation for the theory upon which Monte Carlo methods are based. The pseudo-random number generator in ITS works well as a generator. It does not, however, work very quickly.

When choosing a pseudo-random number generator, besides speed, there are a number of factors to take into consideration: number representation (real or integer), modulus (the range of numbers produced), cycle length (the length of the sequence before repetition) and most importantly – statistical properties (the *randomness* of the numbers). Preferably, a generator is desired that is a single processor instruction, has a modulus the full size that the number representation permits, has an infinite cycle length, and passes all tests for randomness to which it is subjected. This is not deterministically possible for a number of reasons, but as John Von Neumann best said in 1951, “Anyone who considers arithmetical methods of producing random digits is, of course, in a state of sin.” This is why they are called *pseudo*-random number generators. Since a perfect generator can not be found, one that is *good enough* is desired instead.

The Original ITS Random Number Generator: The portable pseudo-random number generator included in ITS was derived from [11]. It has a cycle length of 2^{46} , never generates a zero, and has a single seed value [1]. The main inefficiency identified was that it used double precision real numbers to perform 48-bit integer arithmetic [6]. Since long integers are typically 32-bit, this was probably done to increase the cycle length which would have been limited to about 2^{30} .

For generators which use only one seed value, their cycle lengths have an upper bound equal to the number of values that can be represented by the seed. This is because the next output value of the generator is based solely on the current seed. So to increase the cycle length, the speed of the generator was sacrificed by introducing floating point calculations.

The First Attempt at a New Generator: However, increasing the size of the seed value is not the only way to increase the cycle length. Another alternative is to increase the number of seed values used by the generator. This does require keeping track of more seed values, but it is a small price to pay for speed.

An integer-based generator that uses three seed values was identified in [10]. This generator was only the first half of the suggested generator, but it looked as if it would be sufficient. It has a cycle length of 2^{62} and never produces a zero.

Initial tests using this generator showed that it was nearly four times faster than the original. However, when the new generator was tested in the ITS code, the means and variances of the results did not match those of the original generator. The new generator was simpler than the original in more ways than its arithmetic style.

The first new generator attempted was a simple subtraction-based lagged-Fibonacci generator. It failed some simple tests for linear independence and a statistical test known as the *Birthday-Spacings Test* which has the following algorithm.

Let the [random number generator] produce integers $I_1, I_2, \dots, I_m$ in the range 1 to n . The I 's are m birthdays in a year of n days. The famous problem of Von-Mises and Feller on duplicate birthdays is not stringent enough for a test here, but the problem of duplicate spacings is: Sort the I 's to get $I_{(1)} \leq I_{(2)} \leq \dots \leq I_{(m)}$, and let Y be the number of values which appear more than once among the spacings, $I_{(2)} - I_{(1)}, I_{(3)} - I_{(2)}, \dots, I_{(m-2)} - I_{(m-1)}, I_{(1)} + m - I_{(m)}$. Thus Y is m minus the number of distinct spacings, and Y is asymptotically Poisson with parameter $\lambda = m^3/(4n)$

The birthday spacings test addresses only the set of integers produced by a [random number generator], not the order in which they are produced. Congruential generators generally pass the test. Shift-register generators fail it, as do lagged Fibonacci generators using [addition], [subtraction] or [exclusive-or]. Of the simple generators, only congruential and lagged Fibonacci using multiplication pass the birthday spacings test; most combination generators pass it. [12]

A more statistically robust generator was needed.

The Second Attempt at a New Generator: Combination generators (two or more simple generators put together) are generally more statistically trustworthy than their constituent parts. Therefore, the full implementation of the suggested generator from [10] was attempted next. It has four seed values, produces no zeros, has a cycle length of $2^{32}(p^2+p+1) > 2^{94}$ (with $p=2^{31}-69$), and is integer based.

While the combination generator was slower than its first half, it is still significantly faster than the original generator of ITS. When the combination generator was tested inside the ITS code, the means and variances of the results closely matched those produced with the original generator. The combination generator was successful where the subtraction-based lagged-Fibonacci generator was not.

The only short-coming of the new generator is that it used four seed values instead of one. Significant recoding was required to adapt the input/output functions involving the random number seeds to maintain the dump/restart feature of ITS. The dummy value that was being passed to the original generator (and never used) was removed from all the calling instances of the generator. This removed unnecessary program stack operations which were slowing down the code.

3) *Memory Requirements*: ITS must keep track of a large amount of data while it is simulating radiation transport. Just how much data is not known until run time when the member code reads the input files that define the problem it is to simulate. This would not be a problem if ITS had been written in a computer language that supported dynamic memory allocation. However, at present, ITS is written in FORTRAN 77 and so all of its memory needs must be allocated when the program is compiled.

This means that ITS must overestimate the needs for the average case. Since most modern operating systems use concepts such as multitasking and virtual memory, this over-allocation is undesirable.

Along with the ability to inform the user of what to do in the event of insufficient memory allocation, ITS reports to the user in the output file the sizes of the different arrays used by the program and what portions of those arrays the program actually needed. So as originally written, ITS informs the user of how to minimize the memory requirements for any particular input case.

A method was developed to automate this process. The ITS source code was stripped of everything but its problem initialization routines which is when it determines how much memory it has allocated and how much memory it actually needs. Then the output of this program was reformatted so that it generated update directives for UPEML. These directives could then be applied in addition to the general acceleration directives to the ITS source code to produce a specialized version of the desired member code. This specialized version would be specifically designed to meet the memory needs of the input case it is to simulate.

Unfortunately, this adds a new level of complexity to running a simulation. Each simulation requires a separate compilation to produce its own executable before it can be

run efficiently. However, another use was found for this auxiliary program when the 3-D geometry issues of the ACCEPT member code were investigated.

4) *3-D Geometry*: ACCEPT handles electron/photon tracking in a way very similar to the method used by 3-D computer graphics programs that realistically depict visual scenes. This process is called ray tracing and is very computationally intensive. The discussion presented here will be very general; a more detailed explanation of ray tracing can be found in [13] and a good bibliography of academic papers on the subject can be found in [14].

In ray tracing, a geometric ray is projected from a known point in 3-D space. In ACCEPT this point is the current location of the electron or photon being followed and the ray is the line-of-trajectory of that particle. The ray is tested for intersections with bodies defined in the scene. It must test for intersections with all of the bodies to determine with which body it will intersect first. From there the particle can interact with the body depending on what the body is. But usually at this point the ray is traced again from a new starting point and direction until something happens to cause the process to stop.

As more objects are added to the 3-D scene that defines the problem, the longer ACCEPT must spend checking for intersections, and the more complex the individual object is the longer it takes to trace. (A sphere is much less computationally intensive than a cylinder to check for intersections.)

This is a very important set of routines to encode as efficiently as possible. The authors of ITS seemed to be aware of this because only minuscule improvements could be made to these routines and none of them were actually involved in the ray tracing process. Some debugging checks were left in so they were removed. A check to make sure ACCEPT was capable of handling the geometries of the bodies in the scene was removed since the same check was performed when the scene was read during input and caused the program to abort if any problems were encountered.

However, it was realized that the scene definition that was part of the test case for ACCEPT was redundantly and inefficiently encoded. Composite bodies (or code zones as they are called in ACCEPT) can be constructed from simple geometric shapes like spheres, rectangular boxes, cylinders, etc. by taking positive and negative combinations of their

unions and intersections with each other. A hollow sphere is defined as what is left of a solid sphere when its intersection with a smaller, concentric sphere is subtracted from it. The test case was also subtracting from the outer sphere the four or five rectangular boxes that were completely contained within the inner sphere. This needlessly added the rectangular boxes to the hollow sphere's definition. This and other similar redundancies were causing ACCEPT to spend time checking for intersections that would never be the closest intersection.

Two methods of determining if the scene geometries are encoded efficiently were developed. One method used a Monte Carlo approach while the other was deterministic.

Monte Carlo Geometry Efficiency Verification: A set of update directives were developed to add a hit and miss table for intersection attempts to ACCEPT's output file. The number of particle histories was automatically set to 5000 so that no modification of the ACCEPT input file being tested was necessary. By examining the data in this table, the user can find clues to *possible* inefficiencies in the geometry coding. A body that is in a code zone and is often intersected with but never chosen as the closest intersection could be unnecessary.

This method is not precise. It may expose some inefficiencies and miss others. But it does so regardless of which geometric primitives were used to construct a code zone. It is a helpful tool and should be treated as such. Its update directives can be found in Appendix D.

Deterministic Geometry Efficiency Verification: There are three different types of geometric inefficiencies that can be encoded in the scene definition in ACCEPT that were addressed: boxes (BOX) that should be defined as rectangular parallelepipeds (RPP), negative bodies completely contained within another negative body in a code zone, and positive bodies completely contained within another positive body in a code zone. Code was written to handle a select group of ACCEPT's geometric primitives for these cases. Since only the input routines need to execute to collect the data necessary to conduct these tests, they were added to the separate program that performed the automatic memory reduction. The geometry checking portion of that program's code is in Appendix E.

Code was written to test for the following primitive cases: BOX inside sphere (SPH), BOX inside RPP, BOX inside right circular cylinder (RCC), RPP inside SPH, RPP inside RPP, RPP inside RCC, SPH inside SPH, SPH inside RPP, SPH inside RCC, RCC inside SPH, RCC inside RPP, and RCC inside RCC. If the program finds any such inefficiencies in any code zone definition or any BOXes that should have been coded as RPPes, it reports it to the user as comments at the top of the update directive file produced by the automatic memory reduction portion of the code. It is very specific and reports exactly which bodies are unnecessary in which code zones and directs the user to take the necessary steps to rectify the problem. However, it only works for a limited number of the ACCEPT geometrical primitives and ignores all of the others.

D. Benchmarking the Improved Code

The three different member codes of ITS (TIGER, CYLTRAN and ACCEPT) were again timed using 800,000 particle history versions of each of their individual test cases. This time the new and improved versions of the codes were used including a more efficient version of the ACCEPT test case. The update directives for the improved code can be found in Appendix B and the improved version of the ACCEPT test case can be found in Appendix C. As before, this benchmarking was performed on a Sun SPARCstation 10 running under Solaris 2.4 (SunOS 5.4). The codes were compiled using the Sun f77 FORTRAN compiler with full optimization (-O4 -dalign -native). The times are as shown in Table III.

TABLE III
BENCHMARK TIMES OF MODIFIED ITS MEMBER CODES
ALL TIMES IN HOURS:MINUTES:SECONDS

TIGER	CYLTRAN	ACCEPT
3:23:05.40	3:06:15.38	2:18:27.33

IV. RESULTS

From the 800,000 history benchmark timings shown in Chapter 3, the following speed up factors (original time divided by improved time) were applied to the ITS member codes for each of their individual test cases. The TIGER test case achieved a speed up of 2.00, the CYLTRAN test case achieved a speed up of 1.74, and the ACCEPT test case achieved a speed up of 1.90.

With the exception of the new pseudo-random number generator, none of the modifications made to the ITS member codes changed the results of the simulations in any way. The new pseudo-random number generator produced results with means and variances that closely matched those of the original generator of ITS. Identical results were not expected for this improvement and this statistical similarity in the results made this improvement acceptable.

The speed ups supplied by each of the individual methods used were not equal. Table IV shows the effect of each of the individual improvements over the original. For these improvement testing trials, 100,000 particle history versions of the test cases were used. As in the benchmark timings, they were performed on a Sun SPARCstation 10 running under Solaris 2.4 (SunOS 5.4). Each modified version of the code plus the original was compiled with the f77 FORTRAN compiler once with standard optimization (-O) and once with full optimization (-O4 -dalign -native). For comparative timing purposes, all of the versions of a particular member code of similar compiler optimization were run simultaneously on the same machine. This way any other programs running on the machine at the time would affect the different versions of the ITS code in the same way. The reason for the two different compiler settings should be evident in the ACCEPT timing results.

TABLE IV
INDIVIDUAL METHOD SPEEDUPS IN THE ITS MEMBER CODES
STANDARD OPTIMIZATION WITH -O, FULL OPTIMIZATION WITH -O4 -DALIGN -NATIVE.
ALL TIMES IN HOURS:MINUTES:SECONDS

TIGER	Standard Timing	Standard Speed Up	Full Timing	Full Speed Up
Original	0:53:46.99	1.00	0:51:05.45	1.00
Quadratic Interpolation Routine	0:46:43.58	1.15	0:44:50.45	1.14
Random Number Generator	0:44:59.54	1.20	0:33:48.23	1.51
Memory Requirements	0:52:54.95	1.02	0:49:34.21	1.03
All Methods	0:36:19.03	1.48	0:25:50.15	1.98
CYLTRAN	Standard Timing	Standard Speed Up	Full Timing	Full Speed Up
Original	0:47:50.43	1.00	0:40:33.23	1.00
Quadratic Interpolation Routine	0:42:55.00	1.11	0:36:08.14	1.12
Random Number Generator	0:38:29.60	1.24	0:28:36.01	1.42
Memory Requirements	0:44:41.59	1.07	0:39:38.11	1.02
All Methods	0:32:35.07	1.47	0:23:44.76	1.71
ACCEPT	Standard Timing	Standard Speed Up	Full Timing	Full Speed Up
Original	0:37:44.84	1.00	0:33:00.63	1.00
Quadratic Interpolation Routine	0:36:02.04	1.05	0:33:16.82	0.99
Random Number Generator	0:32:18.28	1.17	0:28:47.23	1.15
Memory Requirements	0:36:30.40	1.03	0:32:17.26	1.02
3-D Geometry	0:24:17.78	1.55	0:20:57.37	1.58
All Methods	0:19:32.71	1.93	0:17:21.10	1.90

A. Additional Benefits

The memory reduction procedure reduced the executable image size of the ACCEPT test case from 4388 KB to 3448 KB for a total savings of 940 KB or 21% (the TIGER case had memory savings of nearly 47%). While this memory reduction has the negative effect of adding more steps into the user process of running a simulation, it has a number of side benefits. Most modern computer operating systems use concepts known as virtual memory (where the hard drive is used as if it were RAM) and multitasking (where multiple programs are run "at the same time"). In such an environment, a smaller required memory allocation for a program allows for a larger percentage of it to be stored in real RAM. This means that less time is spent accessing the hard drive to bring another portion of the program into RAM during execution. This "swap" time varies from one run to the next depending on what other programs are concurrently running on the computer. It is not included in the timing information reported by ITS but is included in the amount of time a user must wait for the program to finish.

In a multitasking environment, smaller programs are sometimes given higher priority than larger programs. In this case, it was observed that when the original version of ITS was run simultaneously against the memory reduced version, the reduced version was allowed to access 10% more of the processor cycles than the original. This means that the reduced version finished well ahead of the original version in real time, even though the internal timing mechanism of ITS reported only marginal improvements in execution time.

The advantages of a smaller program size are not limited to the program itself. Since it must share memory with the other programs running concurrently on the computer, a smaller memory allocation for one program means that there is more memory left over for the other programs. All programs get to store a larger percentage of themselves in real RAM and the same result occurs as did for the smaller program. Hence, the entire computing environment performs better. If the other programs are more ITS simulations, the benefits from this memory reduction technique begin to compound. The memory reduction procedure seems worth the added step to the user process.

B. Negative Effects

Sometimes, to make improvements in speed, sacrifices must be made in other areas. In the case of the acceleration of ITS, extra steps were added to the procedure the user must follow to run a simulation because of the memory reduction program. This adds a compilation step for every simulation which could nullify the time it was supposed to save! It is therefore left to the discretion of the user whether or not this particular speed improvement should be used.

Another negative effect can be noticed in Table IV for the quadratic interpolation routine running under full optimization in ACCEPT. In this one case, the modifications to the search algorithm seem to be slowing down the simulation instead of speeding it up. This is why the tests were also performed with standard optimization. In that case, the modifications did speed up ACCEPT although not as much as in TIGER or CYLTRAN. It is expected that this negative effect is attributable to the compiler being used and not to ITS or to the modifications themselves. Since the search algorithm and interpolation formula modifications are theoretically sound, these enhancements should be used despite this small set back.

V. CONCLUSIONS AND FUTURE WORK

The Integrated TIGER Series of Coupled Electron/Photon Monte Carlo Transport Codes System has been made significantly faster without sacrificing its accuracy. The speed up attained in this study was nearly a factor of 2 in all three member codes. This speed up was the result of more efficient high-level language coding and not that of any assembly level hand coding. Hence this faster version of ITS is just as portable as the original. Since the modifications were made through the use of the UPEML machine-portable CDC update emulator, it is easy to incorporate them into an existing ITS code installation.

A large portion of the speed up seen in the ACCEPT test case was the result of streamlining the geometric description in the input file. As a result, already streamlined input files would not see the performance gains this case did. However, this particular test case for ACCEPT used a photon beam instead of the electron beams used in the TIGER and CYLTRAN cases. Since electron tracking is more computationally expensive than photon tracking, this probably contributed to the ACCEPT case's otherwise low performance gain. Therefore, cases involving only photon interactions could expect to see smaller speed ups than cases involving both electron and photon interactions.

One of the main objectives of the research for this thesis was to adapt ITS into a viable design tool for use at the DECADE pulsed-power facility. With a computational acceleration of nearly 2 and the addition of the geometric efficiency verification routines for ACCEPT, this objective has been accomplished. However, possibilities exist for making ITS an even better design tool.

Subroutines involving the 3-D nature of the ACCEPT codes (specifically those involving ray-tracing) are some of its most computationally expensive. The field of computer graphics should be closely watched for advancements in 3-D scene generation and rendering. Improved algorithms in that field could be applied to the ACCEPT ray tracing routines and result in comparable speed ups in execution time to those achieved by the improved graphics packages.

Parallel processing techniques offer another possible way to decrease the required run time of ITS simulations. By spreading the work out among many processors, a speed up that approaches the number of processors being used could be attained. This can be achieved through the use of a parallel-processing super-computer or a local area network using a parallel computing simulator such as Parallel Virtual Machine (PVM). To keep ITS as portable as possible, the PVM route is probably the best alternative for implementing parallel processing.

Many Monte Carlo based codes have been adapted to run in a parallel environment. The number of histories to be tracked are evenly divided among the processors and either each processor is given a different random seed, or one processor is assigned to act as the random number generator for everyone. Inter-processor communication (for setting up the initial conditions and tabulating the final results) and the random number generation are perhaps the main issues in this implementation. If one processor acts as a dedicated random number generator, there is a degradation in performance due to the extra inter-processor communication. However, if each processor handles its own random number generation, great care must be taken to ensure that no two processors overlap part of the random number sequence.

A more transparent method of applying dynamic memory allocation should be investigated. Dynamic memory allocation is not part of the FORTRAN 77 standard. While most FORTRAN 77 compilers have an extension for supporting this concept, they vary in how they implement this feature and hence cross-platform portability becomes a problem for this option. The FORTRAN 90 standard does contain a method for dynamic memory allocation; however, FORTRAN 90 compilers are not widespread which makes cross-platform portability a problem again. The portability issue could be side-stepped by moving ITS from FORTRAN 77 to a widespread language with a consistent method for dynamic memory allocation such as C or C++. However, shifting to another computer language would be a fairly labor intensive project and possibly not worth the effort.

When viewing ITS as a design tool, one of its weak points is its user interface. A number of programs must be run to complete one simulation. Each program requires its own set of input files in specific places with specific names and if it has an interface, it is

text-based. (ITS is designed to run primarily in the background in which case no interface is desired.) Job control language or shell scripts could be written to automate the process of running a simulation. Portability concerns would need to be addressed, but it is conceivable that scripts could be written for a wide variety of job control languages.

Another method for improving the user interface would be to develop a graphical user interface (GUI) to interact with ITS. GUIs seem to be coming part of the definition of what is user friendly. Developing an interface to give "push button" control over ITS would be easy compared to the modifications that would have to be made to ITS to give the user the type of visual feedback on the progress of the simulation which the user would expect from a GUI. Also, as with many of the other possible improvements already discussed, cross-platform portability is a potential roadblock to making this ease-of-use improvement a reality.

REFERENCES

REFERENCES

- [1] J. A. Halbleib, R. P. Kensek, T. A. Mehlhorn, G. D. Valdez, S. M. Seltzer, and M. J. Berger, "ITS Version 3.0: The Integrated TIGER Series of coupled electron/photon Monte Carlo transport codes," Technical Report SAND91-1634, Sandia National Laboratories, Albuquerque, NM, March 1992.
- [2] J. A. Halbleib, R. P. Kensek, G. D. Valdez, T. A. Mehlhorn, S. M. Seltzer, and M. J. Berger, "ITS Version 3.0: The Integrated TIGER Series of coupled electron/photon Monte Carlo transport codes," *OECD Documents: Advanced Monte Carlo Computer Programs for Radiation Transport*, pp. 47-58, April 1993.
- [3] J. A. Halbleib, R. P. Kensek, G. D. Valdez, S. M. Seltzer, and M. J. Berger, "ITS: The Integrated TIGER Series of electron/photon transport codes - version 3.0," *IEEE Transactions on Nuclear Science*, vol. 39, pp. 1025-1030, April 1992.
- [4] P. Sincerny, S. Ashby, K. Childers, C. Deeney, and L. Schlitt, "The DECADE high power generator," Internal Technical Report, Physics International, San Leandro, CA, 1993.
- [5] A. G. Wooten, "Monte Carlo computations of a pulsed-power brehmsstrahlung source," Master of Science Thesis, The University of Tennessee, Knoxville, TN, December 1993.
- [6] R. D. Hochstedler and L. M. Smith, "Acceleration of a Monte Carlo radiation transport code," *Proceedings, Space Technology & Applications International Forum*, Albuquerque, NM, January 1996.
- [7] J. A. Halbleib, T. W. L. Sanford, and J. W. Poukey, "Radiation environment of Hermes III," *IEEE Transactions on Nuclear Science*, vol. 35, pp. 1282-1287, December 1988.
- [8] D. E. Beutler, J. A. Halbleib, and D. P. Knott, "Comparison of experimental pulse-height distributions in germanium detectors with Integrated TIGER Series code predictions," *IEEE Transactions on Nuclear Science*, vol. 36, pp. 1912-1919, December 1989.
- [9] J. J. Dongarra, "Performace of various computers using standard linear equations software," <http://www.netlib.org/benchmark/performace.ps>, July 1996.
- [10] G. Marsaglia and A. Zaman, "Some portable very-long-period random number generators," *Computers in Physics*, vol. 8, no. 1, pp. 117-121, January/February 1994.
- [11] J. S. Hendricks, "Effects of changing the random number stride in Monte Carlo calculations," *Nucl. Sci. Eng.*, vol. 109, pp. 86-91, September 1991.
- [12] G. Marsaglia, "A current view of random number generators, keynote address," *Proceedings, Computer Science and Statistics: 16th Symposium on the Interface*, pp. 3-10, 1985.
- [13] A. S. Glassner, J. Arvo, R. L. Cook, E. Haines, P. Hanrahan, P. Heckbert and D. B. Kirk, "An introduction to ray tracing," *Academic Press*, 1989.
- [14] I. Grimstead, "The on-line ray tracing bibliography," <http://www.cm.cf.ac.uk/Ray.Tracing/RT.Bibliography.html>, September 1995.

APPENDICES

Appendix A. ITS Test Case Input Files

1) TIGER Input Files

***** XGEN INPUT *****

MATERIAL TA

MATERIAL AL

MATERIAL SI

TITLE

1.0 MEV STANDARD CODES CROSS SECTIONS FOR TA, AL AND SI

ENERGY 1.0

***** TIGER INPUT *****

ECHO 1

TITLE

...1.0 MEV TA/AL/SI FILTERED DETECTOR DECADE TIGER TEST CASE

***** GEOMETRY *****

* MATERIAL SUBZONES THICKNESS ELECTRON-CUTOFF FORCING

GEOMETRY 4

1	1	0.00254
---	---	---------

2	1	0.02540
---	---	---------

2	1	0.20000
---	---	---------

3	1	0.50000
---	---	---------

***** SOURCE *****

ELECTRONS

ENERGY 1.0

CUTOFFS 0.05 0.001

* DEFAULT DIRECTION

DIRECTION 0.0

***** OUTPUT OPTIONS *****

HISTORIES 800000

BATCHES 10

* ... X-RAY PRODUCTION SCALING

SCALE-BREMS 500.

* SCALE-BREMS 5.0

2) CYLTRAN Input Files

***** XGEN INPUT *****

MATERIAL TA

MATERIAL AL

MATERIAL LI 0.2675 F 0.7325

DENSITY 2.635

TITLE

0.3 MEV STANDARD CODES CROSS SECTIONS FOR TA, AL AND LiF

ENERGY 0.3

***** CYLTRAN INPUT *****

TITLE

.....ENERGY DEPOSITION

***** GEOMETRY *****

***MINZ	MAXZ	MINX	MAXX	MAT
---------	------	------	------	-----

GEOMETRY 29

0.0	2.54E-3	0.0	16.51	1
-----	---------	-----	-------	---

2.54E-3	2.79E-2	0.0	16.51	2
---------	---------	-----	-------	---

***VACUUM AND DETECTOR RINGS

2.79E-2	12.7279	0.0	15.24	0
---------	---------	-----	-------	---

12.7279	12.8433	0.0	2.2098	0
---------	---------	-----	--------	---

***RING1

12.7279	12.8433	2.2098	2.2225	2
---------	---------	--------	--------	---

12.7279	12.7406	2.2225	2.8575	2
---------	---------	--------	--------	---

12.8306	12.8433	2.2225	2.8575	2
---------	---------	--------	--------	---

12.7279	12.8433	2.8575	2.8702	2
---------	---------	--------	--------	---

12.7279	12.8433	2.8702	4.7498	0
---------	---------	--------	--------	---

***RING2

12.7279	12.8433	4.7498	4.7625	2
---------	---------	--------	--------	---

12.7279	12.7406	4.7625	5.3975	2
---------	---------	--------	--------	---

12.8306	12.8433	4.7625	5.3975	2
---------	---------	--------	--------	---

12.7279	12.8433	5.3975	5.4102	2
---------	---------	--------	--------	---

12.7279	12.8433	5.4102	9.8298	0
---------	---------	--------	--------	---

***RING3

12.7279	12.8433	9.8298	9.8425	2
---------	---------	--------	--------	---

12.7279	12.7406	9.8425	10.4775	2
---------	---------	--------	---------	---

12.8306	12.8433	9.8425	10.4775	2
---------	---------	--------	---------	---

12.7279	12.8433	10.4775	10.4902	2
---------	---------	---------	---------	---

12.7279	12.8433	10.4902	14.4018	0
---------	---------	---------	---------	---

***RING4

12.7279	12.8433	14.4018	14.4145	2
---------	---------	---------	---------	---

12.7279	12.7406	14.4145	15.0495	2
---------	---------	---------	---------	---

12.7279	12.8433	15.0495	15.0622	2
---------	---------	---------	---------	---

12.8306	12.8433	14.4145	15.0495	2
---------	---------	---------	---------	---

12.7279	12.8433	15.0622	15.2400	0
---------	---------	---------	---------	---

***AL SHIELD

2.79E-2	12.8433	15.2400	16.5100	2
---------	---------	---------	---------	---

***LiF DETECTORS

12.7406	12.8306	2.2225	2.8575	3
---------	---------	--------	--------	---

```

12.7406 12.8306 4.7625 5.3975 3
12.7406 12.8306 9.8425 10.4775 3
12.7406 12.8306 14.4145 15.0495 3
***** SOURCE *****
ELECTRONS
ENERGY 0.3
POSITION      0.0      0.0      0.0
RADIUS        5.08
***** OUTPUT OPTIONS *****
HISTORIES 800000
BATCHES 10
SCALE-BREMS 100
PHOTON-FLUX 24 27
  NBINE 1
  NBINT 1

```

3) ACCEPT Input Files

```

***** XGEN INPUT *****
*/ Parameters
ECHO 1
ENERGY 2.5
MATERIAL CA .6 SI .4
    DENSITY 2.35
MATERIAL FE .94 C .05 CR .005 MN .005
    DENSITY 7.8
MATERIAL PB
    DENSITY 11.34
MATERIAL N .7 O .3
    DENSITY .0013
MATERIAL SI
    DENSITY 2.33
MATERIAL CU
    DENSITY 8.96
MATERIAL AL .166045 CU .604953 C .1962875 H .0327145
    DENSITY 1.27
TITLE
... Test Run, 1.5 Mev photons in 4 ft (122 cm) concrete wall ...
***** To determine hole penetration *****
***** MATERIAL CA .4 SI .575 AL .025 DENSITY 1.75

***** ACCEPT INPUT *****
*/Running Accept model
ECHO 1
TITLE
... Test Run, 1.5 Mev photons in 4 ft (122 cm) concrete wall ...
***** To determine hole penetration of *****
***** Lead and S.S. lined holes *****
***** Geometry *****
GEOMETRY 0.0 0.0
BOX      0.0, 670.56, 0.0, 101.6, 0.0, 0.0,
          0.0, 114.3, 0.0, 0.0, 0.0, 34.29
BOX      .635, 670.56, .635, 100.33, 0.0, 0.0,
          0.0, 114.3, 0.0, 0.0, 0.0, 33.02
BOX      4.445, 670.56, .635, 92.71, 0.0, 0.0,
          0.0, 114.3, 0.0, 0.0, 0.0, 29.21
BOX      5.08, 670.56, .635, 91.44, 0.0, 0.0,
          0.0, 114.3, 0.0, 0.0, 0.0, 28.575
RPP      -670.56, 670.56, 670.56, 784.86, -670.56, 670.56
BOX      5.08, 784.86, .635, 91.44, 0.0, 0.0,
          0.0, 20.0, 0.0, 0.0, 0.0, 28.575
SPH      0.0, 0.0, 0.0, 1200.0
SPH      0.0, 0.0, 0.0, 1201.0
END
***** Zones *****
* Hole
ZSUB001 +4

```

```

10 1 1
* Detector
ZSUB002 +6
2 1 1
* Concrete wall with hole
Z03 +5 -1 -2 -3 -4
* S.S layer outer
Z04 +1 -2 -3 -4
* Lead layer
Z05 +2 -3 -4
* S.S layer inner
Z06 +3 -4
* Surrounding air
Z07 +7 -6 -5 -4 -3 -2 -1
* Escape
Z08 +8 -7 -6 -5 -4 -3 -2 -1
END
7
5
1
2
3
2
4
0
**** Miscellaneous ****
PHOTONS
POSITION 50.08 0.0 177.6984
DIRECTION 90.0 90.0
ISOTROPIC 90.0
HISTORIES 800000
BATCHES 10
ENERGY 1.5
CUTOFFS .04 .01
DUMP

```

Appendix B. UPEML Update Directives

```

*/ -----
*IDENT SPEEDUP
*DELETE,QPOL.00015
C      Last Modified      1995 May 26
*INSERT,QPOL.00021
C      Speedup Enhancement
C      Linear search replaced with binary search.
C      Less computationally intensive algorithm used
C      to compute final quadratic interpolation.
*DELETE,QPOL.00040,QPOL.00123
      if (N .le. 2) then
          T = Y(1) + (S - X(1))*(Y(2) - Y(1))/(X(2) - X(1))
      else
          if (X(N) .ge. X(1)) then
C
C      data is in non-decreasing order
C
          if (S .lt. X(2)) then
              j = 2
          else if (S .gt. X(N - 1)) then
              j = N - 1
          else
              i = 1
              k = N
10          continue
              j = (i + k) / 2
              if (S .gt. X(j)) then
                  i = j
              else
                  k = j
              endif
              if ((k - i) .gt. 1) goto 10
              if (S .lt. (C5EM1 * (X(i - 1) + X(k + 1)))) then
                  j = i
              else
                  j = k
              endif
          endif
      else
C
C      data is in decreasing order
C
          if (S .gt. X(2)) then
              j = 2
          else if (S .lt. X(N - 1)) then
              j = N - 1
          else
              i = 1
              k = N
20          continue
              j = (i + k) / 2
              if (S .lt. X(j)) then

```

```

        i = j
    else
        k = j
    endif
    if ((k - i) .gt. 1) goto 20
    if (S .gt. (C5EM1 * (X(i - 1) + X(k + 1)))) then
        j = i
    else
        j = k
    endif
endif
endif
endif
C
    i = j - 1
    k = j + 1
    T = Y(j) + ((S - X(j)) / (X(k) - X(i))) * (
$      ((S - X(i)) * (Y(k) - Y(j))) / (X(k) - X(j))
$      + ((X(k) - S) * (Y(j) - Y(i))) / (X(j) - X(i)))
    endif
    return
end
*DELETE,BANGLE.00014
C      Last Modified      1995 May 26
*INSERT,BANGLE.00017
C      Speedup Enhancement
C      Linear search replaced with binary search.
*DELETE,BANGLE.00038,BANGLE.00041
    RDJ1 = PKI - RANG(2)
    if (RDJ1 .ge. 0) then
        J1 = 2
        goto 200
    endif
C
    J = 2
    J1 = NRANG
100 continue
    L = (J + J1) / 2
    if (PKI .ge. RANG(L)) then
        J1 = L
    else
        J = L
    endif
    if ((J1 - J) .gt. 1) goto 100
C
    RDJ1 = PKI - RANG(J1)
    if (RDJ1 .ge. 0) goto 200
C
*DELETE,EQDEP.00023
C      Last Modified      1995 May 26
*INSERT,EQDEP.00026
C      Speedup Enhancement
C      Only affects TIGER codes, linear search replaced

```

```

C      with binary search.
*DELETE,EQDEP.00041,EQDEP.00043
      if (ZD .lt. BB(1)) then
        L = 1
        goto 200
      endif
C
      K = 1
      L = NIZON
100  continue
      LTEST = (K + L) / 2
      if (ZD .lt. BB(LTEST)) then
        L = LTEST
      else
        K = LTEST
      endif
      if ((L - K) .gt. 1) goto 100
      if (ZD .lt. BB(L)) goto 200
*DELETE,FOLD.00015
C      Last Modified      1995 December 21
*INSERT,FOLD.00019
C      Speedup Enhancement
C      Computationally intensive bounds check replaced with
C      less computationally intensive method.
*DELETE,FOLD.00037,FOLD.00040
      if (COS1 .gt. CONE) then
        COS1 = CONE
      else
        if (COS1 .lt. -CONE) COS1 = -CONE
      endif
      if (COSD .gt. CONE) then
        COSD = CONE
      else
        if (COSD .lt. -CONE) COSD = -CONE
      endif
C
      COS2 = COS1 * COSD + SQRT((CONE - COS1 * COS1) *
$      (CONE - COSD * COSD)) * CCH(JAZ + 1)
*DELETE,MULT.00014
C      Last Modified      1995 May 26
*INSERT,MULT.00018
C      Speedup Enhancement
C      Linear search replaced with binary search.
*DELETE,MULT.00033
*/ Below is for IF DEF,CDC
*DELETE,MULT.00035
      if (RA .le. G(1,NT)) then
        M = 1
        goto 580
      endif
C
      L = 1

```

```

M = MMAX
570 continue
MTEST = (L + M) / 2
if (RA .le. G(MTEST,NT)) then
    M = MTEST
else
    L = MTEST
endif
if ((M - L) .gt. 1) goto 570
if (RA .le. G(M,NT)) goto 580
*/ Below is for IF -DEF,CDC
*DELETE,MULT.00038
    if (RA .le. G(1,NT,MT)) then
        M = 1
        goto 580
    endif
C
    L = 1
    M = MMAX
570 continue
MTEST = (L + M) / 2
if (RA .le. G(MTEST,NT,MT)) then
    M = MTEST
else
    L = MTEST
endif
if ((M - L) .gt. 1) goto 570
if (RA .le. G(M,NT,MT)) goto 580
*DELETE,MULT.00040
*INSERT,CLASS.00015
C    DECADE Modified          1995 June 21
*INSERT,CLASS.00020
C    DECADE Improvements
C    Replaced linear search algorithm with binary search
*DELETE,CLASS.00041,CLASS.00046
    if (NC .lt. NMAX1) then
        if (TC .lt. EM(NC)) then
            jl = NC - 1
            ju = NMAX1
10        if ((ju - jl) .le. 1) goto 20
            NC = (jl + ju) / 2
            if (TC .lt. EM(NC)) then
                jl = NC
            else
                ju = NC
            end if
            goto 10
20        NC = ju
        end if
    end if
*DELETE,FOLNOK.00010,FOLNOK.00012
    jaz1 = JAZ + 1

```

```

        SIND = SQRT(CONE - COSD * COSD)
        COS2 = COS1 * COSD + SIN1 * SIND * SIGN * CCH(jaz1)
        SIN2 = SQRT(CONE - COS2 * COS2)
*DELETE,FOLNOK.00016,FOLNOK.00017
        CDPH = -SIGN * CCH(jaz1)
        SDPH = SIGN * SCH(jaz1)
*DELETE,FOLNOK.00020
        SDPH = SIND * SIGN * SCH(jaz1) / SIN2
*INSERT,ZONEAA.00014
C        DECADE Modified          1995 June 21
*INSERT,ZONEAA.00018
C        DECADE Improvements
C        Replaced linear search algorithm with binary search
*DELETE,ZONEAA.00032,ZONEAA.00036
        if (SD .gt. DISTX(ICALL,IZC)) goto 10
        jb = 0
        jt = IZC
30      if ((jt - jb) .le. 1) goto 20
        J = (jb + jt) / 2
        if (SD .gt. DISTX(ICALL,J)) then
            jb = J
        else
            jt = J
        end if
        goto 30
*DELETE,ZONEAA.00046
20      J = jt
        LBY = LBX(ICALL,J)
        LBCY = LBCX(ICALL,J)
        return
*/ -----
*/ -----
*IDENT GSPDUP
*/ -- Remove redundant error checking that was done at startup
*DELETE,GG.00088,GG.00096
*/ -- Remove debugging code, this is a release version
*DELETE,DISTA.00055,DISTA.00057
*DELETE,DISTA.00067,DISTA.00069
*DELETE,DISTA.00182,DISTA.00184
*DELETE,DISTA.00188,DISTA.00190
*DELETE,DISTA.00200,DISTA.00202
*DELETE,DISTA.00219,DISTA.00221
*DELETE,DISTA.00235,DISTA.00237
*DELETE,DISTA.00308,DISTA.00310
*DELETE,ZONEA.00043,ZONEA.00045
*DELETE,ZONEA.00055,ZONEA.00057
*DELETE,ZONEA.00071,ZONEA.00073
*DELETE,ZONEA.00083,ZONEA.00085
*DELETE,ZONEA.00135,ZONEA.00137
*DELETE,ZONEF.00045,ZONEF.00047
*DELETE,ZONEF.00062,ZONEF.00064
*DELETE,ZONEF.00068,ZONEF.00070

```

```

*DELETE, JOGEN.00300, JOGEN.00307
*DELETE, JOGEN.00802, JOGEN.00808
*/ -----
*IDENT MZRAND
*/ Optional faster (interger based) random number generator
*/ Replace appropriate random routine with this one if you want
*/ Adapted from mrand suggested random number generator in the
*/ George Marsaglia and Arif Zaman "Some portable very-long-period
*/ random number generator" article in _Computers_in_Physics_, Vol. 8,
*/ No. 1, Jan/Feb 1994 page 117-121
*DELETE, RANVAX.00035, RANVAX.00045

```

C

```

ISEED = ISEED3 - ISEED1
IF(ISEED .LE. 0) ISEED = ISEED + 2147483579
ISEED3 = ISEED2
ISEED2 = ISEED1
ISEED1 = ISEED
ISEED4 = 69069 * ISEED4 + 1013904243
ISEED = ISEED + ISEED4
RAN = 0.5 + 2.328306D-010*DFLOAT(ISEED)
NRN = NRN + 1
RETURN
END

```

C

```

*DELETE, OUT.00013
      INTEGER IRA1, IRA2, IRA3, IRA4
      INTEGER INRAN1, INRAN2, INRAN3, INRAN4
      INTEGER IRC1, IRC2, IRC3, IRC4
*DELETE, OUT.00045
      $      INRAN1, INRAN2, INRAN3, INRAN4,
      $      IRA1, IRA2, IRA3, IRA4,
      $      IRC1, IRC2, IRC3, IRC4, IRNMAX,
*DELETE, STTS.00010
      INTEGER IRSV1, IRSV2, IRSV3, IRSV4
*DELETE, STTS.00018
      $      , IHIST, IRSV1, IRSV2, IRSV3, IRSV4, KPUT, FLMC
*DELETE, RANINT.00002
      SUBROUTINE RANINT(ISEEDP1, ISEEDP2, ISEEDP3, ISEEDP4)
*DELETE, RANINT.00095
      INTEGER ISEEDP1, ISEEDP2, ISEEDP3, ISEEDP4
*DELETE, RANINT.00100, RANINT.00103
      IF (ISEEDP1 .le. 0) then
        ISEED1 = 521288629
        ISEED2 = 362436069
        ISEED3 = 16163801
        ISEED4 = 1131199299
      else
        ISEED1 = ISEEDP1
        ISEED2 = ISEEDP2
        ISEED3 = ISEEDP3
        ISEED4 = ISEEDP4
      endif
endif

```

```

*DELETE,RANINT.00129
    ENTRY RANSAV(ISEEDP1, ISEEDP2, ISEEDP3, ISEEDP4)
    ISEEDP1 = ISEED1
    ISEEDP2 = ISEED2
    ISEEDP3 = ISEED3
    ISEEDP4 = ISEED4
*DELETE,HIST.00109
    CALL RANSAV(IRSAV1, IRSAV2, IRSAV3, IRSAV4)
*DELETE,HIST.00315
    CALL RANSAV(IRC1, IRC2, IRC3, IRC4)
*DELETE,RAN1.00003
    COMMON /RAN1/ ISEED1, ISEED2, ISEED3, ISEED4, NRN
*DELETE,INPUT.00165
    INRAN1 = 521288629
    INRAN2 = 362436069
    INRAN3 = 16163801
    INRAN4 = 1131199299
*DELETE,INPUT.01462,INPUT.01463
    READ (IIN,10000) INRAN1, INRAN2, INRAN3, INRAN4
    IF (IECHO. EQ. 1) WRITE(IOUT,10010) INRAN1,INRAN2,
$      INRAN3,INRAN4
*DELETE,INPUT.01479
    280    READ (ITP14,ERR=9037,END=9038) IBT,INRAN1,INRAN2,INRAN3,
$      INRAN4,BOLD,KPUTMX,IMXOLD
*DELETE,INPUT.01487
    WRITE(IOUT,10010) INRAN1, INRAN2, INRAN3, INRAN4
*DELETE,INPUT.01831,INPUT.01832
10000  FORMAT (I12, I12, I12, I12)
10010  FORMAT (' THE RESTART RANDOM NUMBERS ARE ',I12,I12,I12,I12)
*DELETE,PREP.00314
    IRC1 = INRAN1
    IRC2 = INRAN2
    IRC3 = INRAN3
    IRC4 = INRAN4
*DELETE,HIST.00101
    IF (IB .EQ. 1) then
        INRAN1 = IRA1
        INRAN2 = IRA2
        INRAN3 = IRA3
        INRAN4 = IRA4
    endif
*DELETE,OUTPUT.00171
    10    FORMAT(2(A,T52,4I12/),A,T52,4I12)
*DELETE,OUTPUT.00228,OUTPUT.00230
    WRITE(NPRT,10)' THE INITIAL RANDOM SEEDS OF THIS RUN ARE',
$      INRAN1, INRAN2, INRAN3, INRAN4,
$      ' THE INITIAL RANDOM SEEDS OF THE CURRENT BATCH ARE',
$      IRA1, IRA2, IRA3, IRA4,
$      ' THE FINAL RANDOM SEEDS OF THE CURRENT BARCH ARE',
$      IRC1, IRC2, IRC3, IRC4
*DELETE,PREP.00315
    IRA1 = IRC1

```

```

        IRA2 = IRC2
        IRA3 = IRC3
        IRA4 = IRC4
*DELETE,HIST.00097
        CALL RANINT(IRA1, IRA2, IRA3, IRA4)
*DELETE,OUTPUT.00771
        IRA1 = IRC1
        IRA2 = IRC2
        IRA3 = IRC3
        IRA4 = IRC4
*DELETE,ITS.00323
        WRITE(ITP10) IB,IRC1,IRC2,IRC3,IRC4,BATCH,KPUTMX,IMAX
*DELETE,ABORTX.00041
        IF (FLMC) WRITE(IOUT,9020) IHIST, IB, IRSAV1, IRSAV2,
$          IRSAV3, IRSAV4
*DELETE,ABORTX.00107
$ ' THE INITIAL RANDOM NUMBER SEEDS OF THIS HISTORY ARE ',
$   I12, I12, I12, I12//)
*DELETE,SUBIND.00062
        WRITE(IOUT,1000) ' SUBIND1 -- ',IHIST,IB,IRSAV1,IRSAV2,
$          IRSAV3,IRSAV4
        WRITE(IOUT,1000) ' SUBIND2 -- ',IHIST,IB,IRSAV1,IRSAV2,
$          IRSAV3,IRSAV4
        WRITE(IOUT,1000) ' SUBIND3 -- ',IHIST,IB,IRSAV1,IRSAV2,
$          IRSAV3,IRSAV4
        WRITE(IOUT,1000) ' SUBIND4 -- ',IHIST,IB,IRSAV1,IRSAV2,
$          IRSAV3,IRSAV4
*DELETE,SUBIND.00191
        1000 FORMAT(A,2I10,1X,I12,I12,I12,I12)
*DELETE,HIST.00116
        RA = RAN()
*DELETE,HIST.00141
        RA = RAN()
*DELETE,HIST.00144
        RA = RAN()
*DELETE,HIST.00160
        RA = RAN()
*DELETE,HIST.0199
        RA = RAN()
*DELETE,HIST.00201
        RA = RAN()
*DELETE,EHIST.00155
        RA = RAN()
*DELETE,EHIST.00191
        110 RA = RAN()
*DELETE,EHIST.00549
        52 RAT = RAN()
*DELETE,EHIST.00794
        RA = RAN()
*DELETE,EHIST.00822
        RA = RAN()
*DELETE,EHIST.00895

```

```

      RA = RAN()
*DELETE, SECOND.00040
      RA = RAN()
*DELETE, SECOND.00050
  10      RA = RAN()
*DELETE, SECOND.00052
      RA = RAN()
*DELETE, SECOND.00057
  25      RA = RAN()
*DELETE, SECOND.00059
      RA = RAN()
*DELETE, SECOND.00064
      RA = RAN()
*DELETE, SECOND.00069
      RA = RAN()
*DELETE, LOSS.00042
  10      RA = RAN()
*DELETE, LOSS.00044
      RA = RAN()
*DELETE, LOSS.00048
      RA = RAN()
*DELETE, LOSS.00057
      RA = RAN()
*DELETE, MULT.00032
      RA = RAN()
*DELETE, KNOCK.00038
      RAS = RAN()
*DELETE, KNOCK.00040
      RA = RAN()
*DELETE, KNOCK.00049
  100     RA = RAN()
*DELETE, KNOCK.00051
      RA = RAN()
*DELETE, KNOCK.00056
  200     RA = RAN()
*DELETE, KNOCK.00058
      RA = RAN()
*DELETE, KNOCK.00063
      RA = RAN()
*DELETE, KNOCK.00067
      RA = RAN()
*DELETE, KNOCK.00069
      RA = RAN()
*DELETE, KNOCK.00177
      RAS = RAN()
*DELETE, KNOCK.00179
      RAP = RAN()
*DELETE, KNOCK.00199
      RA = RAN()
*DELETE, KNOCK.00209
  100     RA = RAN()
*DELETE, KNOCK.00211

```

```

        RA = RAN()
*DELETE,KNOCK.00216
    200    RA = RAN()
*DELETE,KNOCK.00218
        RA = RAN()
*DELETE,KNOCK.00223
        RA = RAN()
*DELETE,KNOCK.00228
        RA = RAN()
*DELETE,BREMS.00042
        RAS = RAN()
*DELETE,BREMS.00084
        RAS = RAN()
*DELETE,BREMS.00168
        RAS = RAN()
*DELETE,BREMS.00170
        RAP = RAN()
*DELETE,BREMS.00230
        RAS = RAN()
*DELETE,BREMS.00232
        RAP = RAN()
*DELETE,PHOTS.00034
        RA = RAN()
*DELETE,XRAY.00038
        RAS = RAN()
*DELETE,XRAY.00087
        RAS = RAN()
*DELETE,XRAY.00090
        RAP = RAN()
*DELETE,XRAY.00185
        RAS = RAN() - CONE
*DELETE,XRAY.00191
        RA = RAN()
*DELETE,XRAY.00218
        RAP = RAN()
*DELETE,PSTART.00033
        RA = RAN()
*DELETE,ANGLE.00034
    100    RA = RAN()
*DELETE,ANGLE.00109
    10     RA = RAN()
*DELETE,PHIST.00067
        RA = RAN()
*DELETE,PHIST.00114
        RA = RAN()
*DELETE,PHIST.00143
        RA = RAN()
*DELETE,PHIST.00183
        RA = RAN()
*DELETE,PHIST.00197
        RA = RAN()
*DELETE,PHIST.00249

```

```

830  RA = RAN()
*DELETE,PHIST.00282
      RA = RAN()
*DELETE,PHIST.00351
      RA = RAN()
*DELETE,PHIST.00382
      RA      = RAN()
*DELETE,PHIST.00425
      RA      = RAN()
*DELETE,PHIST.00477
160      RA = RAN()
*DELETE,PHIST.00509
      RA      = RAN()
*DELETE,PHIST.00585
      RA      = RAN()
*DELETE,PHIST.00592
      RA      = RAN()
*DELETE,PHIST.00622
      RA      = RAN()
*DELETE,PHIST.00644
      RA      = RAN()
*DELETE,PHIST.00680
      RA      = RAN()
*DELETE,PHIST.00732
      RA      = RAN()
*DELETE,PHIST.00832
      RA      = RAN()
*DELETE,PHIST.00888
      RA      = RAN()
*DELETE,PHIST.00916
      RA      = RAN()
*DELETE,PHIST.00957
      RA      = RAN()
*DELETE,PHIST.00995
      RA      = RAN()
*DELETE,PHIST.01080
      RA      = RAN()
*DELETE,PHIST.01100
      RA      = RAN()
*DELETE,PHIST.01228
830      RA = RAN()
*DELETE,PHIST.01259
      RA      = RAN()
*DELETE,PHIST.01291
      RA      = RAN()
*DELETE,PHIST.01352
      RA      = RAN()
*DELETE,PHIST.01421
160      RA = RAN()
*DELETE,PHIST.01457
      RA      = RAN()
*DELETE,PHIST.01555

```

```

      RA = RAN()
*DELETE,PHIST.01562
      RA = RAN()
*DELETE,PHIST.01615
      RA = RAN()
*DELETE,PEAUF.00079
  6      IF ((RAN() - PGK(IG)) .GT. CZERO) THEN
*DELETE,PEAUF.00084
  7      IF ((RAN() - PGL1(IG)) .GT. CZERO) THEN
*DELETE,PEAUF.00089
  8      IF ((RAN() - PGL2(IG)) .GT. CZERO) THEN
*DELETE,PEAUF.00094
  9      IF ((RAN() - PGL3(IG)) .GT. CZERO) THEN
*DELETE,PEAUF.00099
 10      IF ((RAN() - PGM(IG)) .GT. CZERO) THEN
*DELETE,KSHLL.00034
      IF ((RAN() - FEK(IG)) .LE. CZERO) THEN
*DELETE,KSHLL.00036
      CPROB = RAN() - FKL3(IG)
*DELETE,KSHLL.00076
      CPROB = RAN() - AKLL(IG)
*DELETE,KSHLL.00079
      CPROB = RAN() - AKL1L1(IG)
*DELETE,KSHLL.00154
      IF ((RAN() - AKLM(IG)) .LE. CZERO) THEN
*DELETE,KSHLL.00156
      CPROB = RAN() - AKL1M(IG)
*DELETE,KSHLL.00194
      CPROB = RAN() - AKL1N(IG)
*DELETE,KSHLL.00231
      CPROB = RAN() - CP91
*DELETE,L1SHLL.00033
      R1 = RAN()
*DELETE,L2SHLL.00033
      R1 = RAN()
*DELETE,L3SHLL.00042
      R1 = RAN()
*DELETE,LSHLL.00017
      IF ((RAN() - WJM) .LE. CZERO) THEN
*DELETE,LSHLL.00036
      R1 = RAN()
*DELETE,MSHLL.00037
      IF ((RAN() - FEM(IG)) .LE. CZERO) THEN
*DELETE,STOPH.00068
      RA = RAN()
*DELETE,STOEL.00101
      RA = RAN()
*DELETE,PEC.00042
  2002      R1 = RAN()
*DELETE,PEC.00045
      IF ((RAN() - XB) .GT. CZERO) GO TO 2002
*DELETE,PEC.00053

```

```

2101      R1 = RAN()
*DELETE,PEC.00056
      IF ((RAN()-P) .GT. CZERO) GO TO 2101
*DELETE,PEC.00060
      IF ((RAN()-P1) .GT. CZERO) GO TO 2101
*DELETE,FLUXG.00149
      FR = RAN()*TRACK
*DELETE,BANGLE.00032
      RA = RAN()
*DELETE,PEAL.00038
20      RAS = RAN()
*DELETE,PEAL.00039
      RA = RAN()
*DELETE,PEAL.00045
      RA = RAN()
*DELETE,SPLIT.00049
      RA = RAN()
*DELETE,SPLIT.00056
      RA = RAN()
*DELETE,SCATT.00031
100     RA = RAN()
*DELETE,SCATT.00033
      RA = RAN()
*DELETE,SCATT.00035
      RA = RAN()
*DELETE,SCATT.00038
      RA = RAN()
*DELETE,SCATT.00040
      RA = RAN()
*DELETE,INCOH.00060
10      RA = RAN()
*DELETE,INCOH.00062
      RA = RAN()
*DELETE,INCOH.00064
      RA = RAN()
*DELETE,INCOH.00067
      RA = RAN()
*DELETE,INCOH.00069
      RA = RAN()
*DELETE,INCOH.00074
      RA = RAN()
*DELETE,INCOH.00076
      RA = RAN()
*DELETE,INCOH.00106
      RA = RAN()
*DELETE,INCOH.00126
150     RA = RAN()
*DELETE,INCOH.00137
      RA = RAN()
*DELETE,COH.00051
10      RA = RAN()
*DELETE,COH.00053

```

```

      RA = RAN()
*DELETE,COH.00079
      RA = RAN()
*DELETE,COH.00109
  130      RA = RAN()
*DELETE,COH.00125
      RA = RAN()
*DELETE,FOLD.00032
      RA = RAN()
*DELETE,FOLD.00077
      RA = RAN()
*DELETE,KICK.00046
  200      RAS = RAN()
*DELETE,KICK.00047
      RA = RAN()
*DELETE,KICK.00056
      RA = RAN()
*DELETE,KICK.00072
  500      RAS = RAN()
*DELETE,KICK.00073
      RA = RAN()
*DELETE,KICK.00088
      RA = RAN()
*DELETE,KICK.00129
  1000     RA = RAN()
*DELETE,KICK.00163
  999      RA = RAN()
*DELETE,KICK.00257
  200      RAS = RAN()
*DELETE,KICK.00258
      RA = RAN()
*DELETE,KICK.00270
      RA = RAN()
*DELETE,KICK.00292
  400      RAS = RAN()
*DELETE,KICK.00293
      RA = RAN()
*DELETE,KICK.00311
      RA = RAN()
*DELETE,KICK.00410
      RA = RAN()
*DELETE,KICK.00466
      RA = RAN()
*DELETE,RANGLE.00032
      RA = RAN()
*DELETE,RANGLE.00035
      RA = RAN()
*/ -----

```

Appendix C. Improved ACCEPT Test Case Input File

```

***** ACCEPT INPUT *****
*/Running Accept model
ECHO 1
TITLE
... Test Run, 1.5 Mev photons in 4 ft (122 cm) concrete wall ...
***** To determine hole penetration of *****
***** Lead and S.S. lined holes *****
***** Geometry *****
GEOMETRY 0.0 0.0
RPP      0.000, 101.600, 670.560, 784.860, 0.000, 34.290
RPP      0.635, 100.965, 670.560, 784.860, 0.635, 33.655
RPP      4.445, 97.155, 670.560, 784.860, 0.635, 29.845
RPP      5.080, 96.520, 670.560, 784.860, 0.635, 29.210
RPP     -670.560, 670.560, 670.560, 784.860, -670.560, 670.560
RPP      5.080, 96.520, 784.860, 804.860, 0.635, 29.210
SPH      0.000, 0.000, 0.000, 1200.000
SPH      0.000, 0.000, 0.000, 1201.000
END
***** Zones *****
* Hole
ZSUB001 +4
10 1 1
* Detector
ZSUB002 +6
2 1 1
* Concrete wall with hole
Z03 +5 -1
* S.S layer outer
Z04 +1 -2
* Lead layer
Z05 +2 -3
* S.S layer inner
Z06 +3 -4
* Surrounding air
Z07 +7 -6 -5
* Escape
Z08 +8 -7
END
7
5
1
2
3
2
4
0
**** Miscellaneous ****
PHOTONS
POSITION 50.08 0.0 177.6984
DIRECTION 90.0 90.0
ISOTROPIC 90.0
HISTORIES 800000

```

BATCHES 10
ENERGY 1.5
CUTOFFS .04 .01
DUMP

Appendix D. ACCEPT Monte Carlo Geometry Efficiency Verification

```

*IDENT,ANALYZE
*/ Modifies:      INPUT  (only 5000 histories allowed)
*/              DISTA  (where tabulation occurs)
*/              OUTPUT (result reporting)
*/              COG    (where array is stored)
*/
*INSERT,COG.00003
$      IANALY(ITMA),
*DELETE,INPUT.00902
      IMAX = 5000
      WRITE(IOUT,*) '3-D GEOMETRY ANALYSIS RUN, HISTORIES SET AT 5000'
*INSERT,DISTA.00066
      IF(RIN .LT. PINF) IANALY(I) = IANALY(I) + 1
*INSERT,DISTA.00085
      IANALY(LSAVE+1) = IANALY(LSAVE+1) + 1
*INSERT,OUTPUT.00046
*CALL,GOMLOC
*INSERT,OUTPUT.00768
      WRITE(IOUT,('(//'' 3-D GEOMETRY CODEZONE STATISTICS''))')
      WRITE(IOUT,('( '' BODIES THAT WERE INTERSECTED BUT NOT THE '' ,
$      ''MINIMUM ARE PROBABLY UNNECESSARY.''))')
      DO 1111 N=1,NUMR
        WRITE(IOUT,('( ''      CODE ZONE '' ,I3,':')') N
        J = 0
        DO 2222 M=1,NUMBOD(N)
          WRITE(IOUT,('( ''      BODY '' ,I3, '' INTERSECTED: '' ,
$          I10, '' WAS MINIMUM: '' ,I10)') M,
$          IANALY(LOCREG(N) + 4*M - 4),
$          IANALY(LOCREG(N) + 4*M - 3)
          J = J + IANALY(LOCREG(N) + 4*M - 3)
        2222 CONTINUE
        WRITE(IOUT,('( ''      CODE ZONE WAS TESTED '' ,I10, ''
TIMES.''))')
        $      J
      1111 CONTINUE
*/ -----

```

Appendix E. ACCEPT Deterministic Geometry Efficiency Verification

```

write(IOUT,(' '*/ Checking geometry efficiency'))
C
C here's where we check for user geometry inefficiency
C
do 191 i=1,numr
  do 192 j=1,NUMBOD(i)
C check for should be RPP coding
  do 193 k=1,NUMBOD(i)
    if (k .eq. j) then
C j and k are same body see if it is a BOX that should be an RPP
      lj = LOCREG(i) + ((j - 1) * 4)
      lj = MA(lj + 1)
      obodyj = OTYPE(MA(lj + 1))
      lj = MA(lj + 4)
      if(obodyj .eq. 'BOX') then
        m = 0
        do 194 l=5,13
          if(FPD(lj + 1) .eq. 0.0) m = m + 1
194 continue
          if(m .eq. 6)
$           write(IOUT,(' '*/ CODE ZONE ',i3,
$             ' : BODY # ',i3,
$             ' : BOX should be RPP'))',i,j
        endif
      else
C check to see if j is INSIDE k
        lk = LOCREG(i) + ((k - 1) * 4)
        lj = LOCREG(i) + ((j - 1) * 4)
C check to see if both bodies are + or -
        if(((MA(lj) .lt. 0) .and. (MA(lk) .lt. 0) .or.
$         (MA(lj) .gt. 0) .and. (MA(lk) .gt. 0))) then
          lk = MA(lk + 1)
          lj = MA(lj + 1)
C now have location of of body info in MA array
          obodyk = OTYPE(MA(lk + 1))
          obodyj = OTYPE(MA(lj + 1))
C now have body types
          lk = MA(lk + 4)
          lj = MA(lj + 4)
C now have beginning of body info in FPD array
          if(obodyj .eq. 'BOX') then
C Compute Vertices
            vert(1,1) = FPD(lj + 2)
            vert(1,2) = FPD(lj + 3)
            vert(1,3) = FPD(lj + 4)
            vert(2,1) = vert(1,1) + FPD(lj + 5)
            vert(2,2) = vert(1,2) + FPD(lj + 6)
            vert(2,3) = vert(1,3) + FPD(lj + 7)
            vert(3,1) = vert(1,1) + FPD(lj + 8)
            vert(3,2) = vert(1,2) + FPD(lj + 9)
            vert(3,3) = vert(1,3) + FPD(lj + 10)
            vert(4,1) = vert(1,1) + FPD(lj + 11)

```

```

vert(4,2) = vert(1,2) + FPD(lj + 12)
vert(4,3) = vert(1,3) + FPD(lj + 13)
vert(5,1) = vert(2,1) + FPD(lj + 8)
vert(5,2) = vert(2,2) + FPD(lj + 9)
vert(5,3) = vert(2,3) + FPD(lj + 10)
vert(6,1) = vert(3,1) + FPD(lj + 11)
vert(6,2) = vert(3,2) + FPD(lj + 12)
vert(6,3) = vert(3,3) + FPD(lj + 13)
vert(7,1) = vert(4,1) + FPD(lj + 5)
vert(7,2) = vert(4,2) + FPD(lj + 6)
vert(7,3) = vert(4,3) + FPD(lj + 7)
vert(8,1) = vert(7,1) + FPD(lj + 8)
vert(8,2) = vert(7,2) + FPD(lj + 9)
vert(8,3) = vert(7,3) + FPD(lj + 10)

C      check for BOX in RPP
      if(obodyk .eq. 'RPP') then
        amax = vert(1,1)
        amix = amax
        amay = vert(1,2)
        amiy = amay
        amaz = vert(1,3)
        amiz = amaz
        do 195 m=2,8
          if(amax.lt.vert(m,1)) amax=vert(m,1)
          if(amix.gt.vert(m,1)) amix=vert(m,1)
          if(amay.lt.vert(m,2)) amay=vert(m,2)
          if(amiy.gt.vert(m,2)) amiy=vert(m,2)
          if(amaz.lt.vert(m,3)) amaz=vert(m,3)
          if(amiz.gt.vert(m,3)) amiz=vert(m,3)
195      continue
C      check extents
      if((amax .le. FPD(lk + 3)) .and.
$      (amin .ge. FPD(lk + 2)) .and.
$      (amay .le. FPD(lk + 5)) .and.
$      (amiy .ge. FPD(lk + 4)) .and.
$      (amaz .le. FPD(lk + 7)) .and.
$      (amiz .ge. FPD(lk + 6)))
$      write(IOUT,(' '* CODE ZONE ',i3,
$      ' : BODY # ',i3,' INSIDE BODY # ',
$      i3,' (BOX IN RPP)')) i,j,k

C      check for BOX in SPH
      else if(obodyk.eq.'SPH') then
C      compute distance from vertices to center of sphere
        do 196 m=1,8
          radii(m) = 0.0
          do 197 n=1,3
            radii(m) = radii(m) +
$              ((vert(m,n) - FPD(lk+n+1)) *
$              (vert(m,n) - FPD(lk+n+1)))
197      continue
          radii(m) = sqrt(radii(m))
196      continue

```

```

C      check against sphere radius
      if((radii(1) .le. FPD(1k+5)) .and.
$      (radii(2) .le. FPD(1k+5)) .and.
$      (radii(3) .le. FPD(1k+5)) .and.
$      (radii(4) .le. FPD(1k+5)) .and.
$      (radii(5) .le. FPD(1k+5)) .and.
$      (radii(6) .le. FPD(1k+5)) .and.
$      (radii(7) .le. FPD(1k+5)) .and.
$      (radii(8) .le. FPD(1k+5)))
$      write(IOUT,(''*/ CODE ZONE '' ,i3,
$      ' ' : BODY # '' ,i3, ' ' INSIDE BODY # '' ,
$      i3, ' ' (BOX IN SPH)'')) i,j,k
C      check for BOX in RCC (same as RPP in RCC)
      else if(obodyk .eq. 'RCC') then
      vert(9,1) = FPD(1k+5) + FPD(1k+2)
      vert(9,2) = FPD(1k+6) + FPD(1k+3)
      vert(9,3) = FPD(1k+7) + FPD(1k+4)
      radii(4) = 1.0 / sqrt(FPD(1k+5) *
$      FPD(1k+5) + FPD(1k+6) * FPD(1k+6)
$      + FPD(1k+7) * FPD(1k+7))
      vert(12,1) = FPD(1k+5) * radii(4)
      vert(12,2) = FPD(1k+6) * radii(4)
      vert(12,3) = FPD(1k+7) * radii(4)
      do 203 m=1,8
C      first check for being beyond radius of cylinder
      vert(10,1) = vert(m,1) - FPD(1k+2)
      vert(10,2) = vert(m,2) - FPD(1k+3)
      vert(10,3) = vert(m,3) - FPD(1k+4)
      vert(11,1) = vert(10,2) * FPD(1k+7) -
$      vert(10,3) * FPD(1k+6)
      vert(11,2) = vert(10,3) * FPD(1k+5) -
$      vert(10,1) * FPD(1k+7)
      vert(11,3) = vert(10,1) * FPD(1k+6) -
$      vert(10,2) * FPD(1k+5)
      radii(1) = sqrt(vert(11,1) * vert(11,1)
$      + vert(11,2) * vert(11,2)
$      + vert(11,3) * vert(11,3))
      if(radii(1) * radii(4) .gt. FPD(1k+8))
$      goto 204
C      then check for being beyond top or bottom of cylinder
      radii(5) = vert(10,1) * vert(12,1) +
$      vert(10,2) * vert(12,2) +
$      vert(10,3) * vert(12,3)
C      check for vertex is proper direction from cylinder bottom
      if(radii(5) .lt. 0.0) goto 204
C      check top of cylinder
      if(radii(5) .gt. 1.0 / radii(4)) goto 204
203      continue
      write(IOUT,(''*/ CODE ZONE '' ,i3,
$      ' ' : BODY # '' ,i3, ' ' INSIDE BODY # '' ,
$      i3, ' ' (BOX IN RCC)'')) i,j,k

```

```

204             endif
C      end of BOX in something checks move on to RPP in something checks
           else if(obodyj .eq. 'RPP') then
C      compute vertices of RPP
           vert(1,1) = FPD(lj + 2)
           vert(1,2) = FPD(lj + 3)
           vert(1,3) = FPD(lj + 4)
           vert(2,1) = FPD(lj + 2)
           vert(2,2) = FPD(lj + 3)
           vert(2,3) = FPD(lj + 7)
           vert(3,1) = FPD(lj + 2)
           vert(3,2) = FPD(lj + 6)
           vert(3,3) = FPD(lj + 4)
           vert(4,1) = FPD(lj + 2)
           vert(4,2) = FPD(lj + 6)
           vert(4,3) = FPD(lj + 7)
           vert(5,1) = FPD(lj + 5)
           vert(5,2) = FPD(lj + 3)
           vert(5,3) = FPD(lj + 4)
           vert(6,1) = FPD(lj + 5)
           vert(6,2) = FPD(lj + 3)
           vert(6,3) = FPD(lj + 7)
           vert(7,1) = FPD(lj + 5)
           vert(7,2) = FPD(lj + 6)
           vert(7,3) = FPD(lj + 4)
           vert(8,1) = FPD(lj + 5)
           vert(8,2) = FPD(lj + 6)
           vert(8,3) = FPD(lj + 7)
C      check for RPP in RPP (really easy case)
           if(obodyk .eq. 'RPP') then
               if((FPD(lj + 2) .ge. FPD(lk + 2)) .and.
$                   (FPD(lj + 3) .le. FPD(lk + 3)) .and.
$                   (FPD(lj + 4) .ge. FPD(lk + 4)) .and.
$                   (FPD(lj + 5) .le. FPD(lk + 5)) .and.
$                   (FPD(lj + 6) .ge. FPD(lk + 6)) .and.
$                   (FPD(lj + 7) .ge. FPD(lj + 7)))
$                   write(IOUT,(' '* CODE ZONE ',i3,
$                   ' ' : BODY # ',i3,' INSIDE BODY # ',
$                   i3,' (RPP IN RPP)')) i,j,k
C      check for RPP in SPH (just like BOX in SPH)
           else if(obodyk .eq. 'SPH') then
C      compute distance from vertices to center of sphere
           do 198 m=1,8
               radii(m) = 0.0
               do 199 n=1,3
                   radii(m) = radii(m) +
$                       ((vert(m,n) - FPD(lk+n+1)) *
$                       (vert(m,n) - FPD(lk+n+1)))
199                   continue
                   radii(m) = sqrt(radii(m))
198                   continue
C      check against sphere radius

```

```

        if((radii(1) .le. FPD(1k+5)) .and.
$           (radii(2) .le. FPD(1k+5)) .and.
$           (radii(3) .le. FPD(1k+5)) .and.
$           (radii(4) .le. FPD(1k+5)) .and.
$           (radii(5) .le. FPD(1k+5)) .and.
$           (radii(6) .le. FPD(1k+5)) .and.
$           (radii(7) .le. FPD(1k+5)) .and.
$           (radii(8) .le. FPD(1k+5)))
$           write(IOUT,(' '* / CODE ZONE ',i3,
$           ' ' : BODY # ',i3,' INSIDE BODY # ',
$           i3,' (RPP IN SPH)')) i,j,k
C    check for RPP in RCC (same as BOX in RCC)
        else if(obodyk .eq. 'RCC') then
            vert(9,1) = FPD(1k+5) + FPD(1k+2)
            vert(9,2) = FPD(1k+6) + FPD(1k+3)
            vert(9,3) = FPD(1k+7) + FPD(1k+4)
            radii(4) = 1.0 / sqrt(FPD(1k+5) *
$                FPD(1k+5) + FPD(1k+6) * FPD(1k+6)
$                + FPD(1k+7) * FPD(1k+7))
            vert(12,1) = FPD(1k+5) * radii(4)
            vert(12,2) = FPD(1k+6) * radii(4)
            vert(12,3) = FPD(1k+7) * radii(4)
            do 205 m=1,8
C    first check for being beyond radius of cylinder
                vert(10,1) = vert(m,1) - FPD(1k+2)
                vert(10,2) = vert(m,2) - FPD(1k+3)
                vert(10,3) = vert(m,3) - FPD(1k+4)
                vert(11,1) = vert(10,2) * FPD(1k+7) -
$                    vert(10,3) * FPD(1k+6)
                vert(11,2) = vert(10,3) * FPD(1k+5) -
$                    vert(10,1) * FPD(1k+7)
                vert(11,3) = vert(10,1) * FPD(1k+6) -
$                    vert(10,2) * FPD(1k+5)
                radii(1) = sqrt(vert(11,1) * vert(11,1)
$                    + vert(11,2) * vert(11,2)
$                    + vert(11,3) * vert(11,3))
                if(radii(1) * radii(4) .gt. FPD(1k+8))
$                    goto 206
C    then check for being beyond top or bottom of cylinder
                radii(5) = vert(10,1) * vert(12,1) +
$                    vert(10,2) * vert(12,2) +
$                    vert(10,3) * vert(12,3)
C    check for vertex is proper direction from cylinder bottom
                if(radii(5) .lt. 0.0) goto 206
C    check top of cylinder
                if(radii(5) .gt. 1.0 / radii(4)) goto 206
205            continue
                write(IOUT,(' '* / CODE ZONE ',i3,
$                ' ' : BODY # ',i3,' INSIDE BODY # ',
$                i3,' (RPP IN RCC)')) i,j,k
206            endif
C    end of RPP in something checks move on to SPH in something checks

```

```

                else if(obodyj .eq. 'SPH') then
C      check for SPH in RPP
                if(obodyk .eq. 'RPP') then
C      check center +/- radius against extents
                if((FPD(1j+2) - FPD(1j+5) .ge. FPD(1k + 2)) .and.
$          (FPD(1j+2) + FPD(1j+5) .le. FPD(1k + 3)) .and.
$          (FPD(1j+3) - FPD(1j+5) .ge. FPD(1k + 4)) .and.
$          (FPD(1j+3) + FPD(1j+5) .le. FPD(1k + 5)) .and.
$          (FPD(1j+4) - FPD(1j+5) .ge. FPD(1k + 6)) .and.
$          (FPD(1j+4) + FPD(1j+5) .le. FPD(1k + 7)))
$          write(IOUT,(' '* CODE ZONE ',i3,
$          ' : BODY # ',i3,' INSIDE BODY # ',
$          i3,' (SPH IN RPP)')) i,j,k
C      check for SPH in SPH
                else if(obodyk .eq. 'SPH') then
C      distance between centers + radius1 < radius for inside
                radii(1) = 0.0
                do 201, m=1,3
                    radii(1) = radii(1) +
$          ((FPD(1j+1+m) - FPD(1k+1+m)) *
$          (FPD(1j+1+m) - FPD(1k+1+m)))
201                continue
                radii(1) = sqrt(radii(1))
                if(FPD(1j+5) + radii(1) .le. FPD(1k+5))
$          write(IOUT,(' '* CODE ZONE ',i3,
$          ' : BODY # ',i3,' INSIDE BODY # ',
$          i3,' (SPH IN SPH)')) i,j,k
C      check for SPH in RCC
                else if(obodyk .eq. 'RCC') then
C      store center top point of cylinder in vert(1,?)
                vert(1,1) = FPD(1k+2) + FPD(1k+5)
                vert(1,2) = FPD(1k+3) + FPD(1k+6)
                vert(1,3) = FPD(1k+4) + FPD(1k+7)
C      find distance from sphere center to radius axis
                vert(2,1) = FPD(1j+2) - FPD(1k+2)
                vert(2,2) = FPD(1j+3) - FPD(1k+3)
                vert(2,3) = FPD(1j+4) - FPD(1k+4)
                vert(3,1) = vert(1,1) - FPD(1k+2)
                vert(3,2) = vert(1,2) - FPD(1k+3)
                vert(3,3) = vert(1,3) - FPD(1k+4)
                vert(4,1) = vert(2,2) * vert(3,3)
$          - vert(2,3) * vert(3,2)
                vert(4,2) = vert(2,3) * vert(3,1)
$          - vert(2,1) * vert(3,3)
                vert(4,3) = vert(2,1) * vert(3,2)
$          - vert(2,2) * vert(3,1)
                radii(1) = sqrt(vert(4,1) * vert(4,1) +
$          vert(4,2) * vert(4,2) +
$          vert(4,3) * vert(4,3))
                radii(2) = sqrt(vert(3,1) * vert(3,1) +
$          vert(3,2) * vert(3,2) +
$          vert(3,3) * vert(3,3))

```

```

                                radii(3) = radii(1)/radii(2)
C      compare this distance + sphere radius to cylinder radius
                                if(radii(3) + FPD(1j+5) .gt. FPD(1k+8))
$                                goto 202
C      still possible that sphere is inside cylinder, check top and
bottom
                                vert(5,1) = FPD(1j+2) - FPD(1k+2)
                                vert(5,2) = FPD(1j+3) - FPD(1k+3)
                                vert(5,3) = FPD(1j+4) - FPD(1k+4)
                                radii(4) = 1.0 / sqrt(FPD(1k+5) * FPD(1k+5)
$                                + FPD(1k+6) * FPD(1k+6) +
$                                FPD(1k+7) * FPD(1k+7))
                                vert(6,1) = FPD(1k+5) * radii(4)
                                vert(6,2) = FPD(1k+6) * radii(4)
                                vert(6,3) = FPD(1k+7) * radii(4)
                                radii(5) = vert(5,1) * vert(6,1) +
$                                vert(5,2) * vert(6,2) +
$                                vert(5,3) * vert(6,3)
C      check for sphere center is proper direction from cylinder bottom
                                if(radii(5) .le. 0.0) goto 202
C      check bottom of cylinder
                                if(radii(5) - FPD(1j+5) .lt. 0.0) goto 202
C      check top of cylinder
                                if((radii(5) + FPD(1j+5))
$                                .gt. (1.0 / radii(4))) goto 202
                                write(IOUT,(' '* CODE ZONE ',i3,
$                                ' : BODY # ',i3,' INSIDE BODY # ',
$                                i3,' (SPH IN RCC)')) i,j,k
202                                endif
C      end of SPH in something checks move on to RCC in something checks
                                else if(obodyj .eq. 'RCC') then
C      center of cylinder top
                                vert(1,1) = FPD(1j+2) + FPD(1j+5)
                                vert(1,2) = FPD(1j+3) + FPD(1j+6)
                                vert(1,3) = FPD(1j+4) + FPD(1j+7)
                                radii(1) = 1.0 / sqrt(FPD(1j+5) *
$                                FPD(1j+5) + FPD(1j+6) * FPD(1j+6)
$                                + FPD(1j+7) * FPD(1j+7))
                                if(obodyk .eq. 'RPP') then
C      check for axial alignment which can could cause problems
                                n = 0
                                do 279 m = 5, 7
                                    if (FPD(1j+m) .eq. 0.0) n = n + 1
279                                continue
                                if (n .eq. 2) then
C      axial alignment
                                    if(FPD(1j+5) .ne. 0.0) then
C      axis in x-direction
                                        if((FPD(1j+2) .lt. FPD(1k+2)) .or.
$                                        (FPD(1j+2) .gt. FPD(1k+3)))
$                                        goto 209
                                        if((vert(1,1) .lt. FPD(1k+2)) .or.

```

```

$          (vert(1,1) .gt. FPD(1k+3)))
$          goto 209
$          if((FPD(1j+3) + FPD(1j+8) .gt.
$            FPD(1k+5)) .or. (FPD(1j+3) -
$            FPD(1j+8) .lt. FPD(1k+4)))
$            goto 209
$            if((FPD(1j+4) + FPD(1j+8) .gt.
$              FPD(1k+7)) .or. (FPD(1j+4) -
$              FPD(1j+8) .lt. FPD(1k+6)))
$              goto 209
$          else if(FPD(1j+6) .ne. 0.0) then
C      axis in y-direction
$          if((FPD(1j+3) .lt. FPD(1k+4)) .or.
$            (FPD(1j+3) .gt. FPD(1k+5)))
$            goto 209
$          if((vert(1,2) .lt. FPD(1k+2)) .or.
$            (vert(1,2) .gt. FPD(1k+5)))
$            goto 209
$          if((FPD(1j+2) + FPD(1j+8) .gt.
$            FPD(1k+3)) .or. (FPD(1j+2) -
$            FPD(1j+8) .lt. FPD(1k+2)))
$            goto 209
$          if((FPD(1j+4) + FPD(1j+8) .gt.
$            FPD(1k+7)) .or. (FPD(1j+4) -
$            FPD(1j+8) .lt. FPD(1k+6)))
$            goto 209
$          else
C      axis in z-direction
$          if((FPD(1j+4) .lt. FPD(1k+6)) .or.
$            (FPD(1j+4) .gt. FPD(1k+7)))
$            goto 209
$          if((vert(1,3) .lt. FPD(1k+6)) .or.
$            (vert(1,3) .gt. FPD(1k+7)))
$            goto 209
$          if((FPD(1j+3) + FPD(1j+8) .gt.
$            FPD(1k+5)) .or. (FPD(1j+3) -
$            FPD(1j+8) .lt. FPD(1k+4)))
$            goto 209
$          if((FPD(1j+2) + FPD(1j+8) .gt.
$            FPD(1k+3)) .or. (FPD(1j+2) -
$            FPD(1j+8) .lt. FPD(1k+2)))
$            goto 209
$          endif
$          goto 208
$          endif
$          do 208 m = 1, 6
C      do min/max for each x,y,z of each base by putting specifics
C      into generics
C      vert(2,?) = MIN - base center
C      vert(3,?) = MAX - base center
C      vert(4,?) = base center
C      FPD(1j+8) = base radius

```

```

if(m .eq. 1) then
  vert(2,1) = FPD(1k+2) - FPD(1j+2)
  vert(2,2) = 0.0
  vert(2,3) = 0.0
  vert(3,1) = FPD(1k+3) - FPD(1j+2)
  vert(3,2) = 0.0
  vert(3,3) = 0.0
  vert(4,1) = FPD(1j+2)
  vert(4,2) = FPD(1j+3)
  vert(4,3) = FPD(1j+4)
else if(m .eq. 2) then
  vert(2,1) = 0.0
  vert(2,2) = FPD(1k+4) - FPD(1j+3)
  vert(2,3) = 0.0
  vert(3,1) = 0.0
  vert(3,2) = FPD(1k+5) - FPD(1j+3)
  vert(3,3) = 0.0
  vert(4,1) = FPD(1j+2)
  vert(4,2) = FPD(1j+3)
  vert(4,3) = FPD(1j+4)
else if(m .eq. 3) then
  vert(2,1) = 0.0
  vert(2,2) = 0.0
  vert(2,3) = FPD(1k+6) - FPD(1j+4)
  vert(3,1) = 0.0
  vert(3,2) = 0.0
  vert(3,3) = FPD(1k+7) - FPD(1j+4)
  vert(4,1) = FPD(1j+2)
  vert(4,2) = FPD(1j+3)
  vert(4,3) = FPD(1j+4)
else if(m .eq. 4) then
  vert(2,1) = FPD(1k+2) - vert(1,1)
  vert(2,2) = 0.0
  vert(2,3) = 0.0
  vert(3,1) = FPD(1k+3) - vert(1,1)
  vert(3,2) = 0.0
  vert(3,3) = 0.0
  vert(4,1) = vert(1,1)
  vert(4,2) = vert(1,2)
  vert(4,3) = vert(1,3)
else if(m .eq. 5) then
  vert(2,1) = 0.0
  vert(2,2) = FPD(1k+4) - vert(1,2)
  vert(2,3) = 0.0
  vert(3,1) = 0.0
  vert(3,2) = FPD(1k+5) - vert(1,2)
  vert(3,3) = 0.0
  vert(4,1) = vert(1,1)
  vert(4,2) = vert(1,2)
  vert(4,3) = vert(1,3)
else
  vert(2,1) = 0.0

```

```

vert(2,2) = 0.0
vert(2,3) = FPD(1k+6) - vert(1,3)
vert(3,1) = 0.0
vert(3,2) = 0.0
vert(3,3) = FPD(1k+7) - vert(1,3)
vert(4,1) = vert(1,1)
vert(4,2) = vert(1,2)
vert(4,3) = vert(1,3)
endif

C      check min

$      radii(2) = radii(1) * (vert(2,1) *
$      FPD(1j+5) + vert(2,2) * FPD(1j+6)
$      + vert(2,3) * FPD(1j+7))
vert(5,1) = radii(2) * radii(1) *
$      FPD(1j+5)
vert(5,2) = radii(2) * radii(1) *
$      FPD(1j+6)
vert(5,3) = radii(2) * radii(1) *
$      FPD(1j+7)
vert(6,1) = vert(2,1) - vert(5,1)
vert(6,2) = vert(2,2) - vert(5,2)
vert(6,3) = vert(2,3) - vert(5,3)
radii(3) = sqrt(vert(6,1) *
$      vert(6,1) + vert(6,2) * vert(6,2)
$      + vert(6,3) * vert(6,3))
if(radii(3) .ne. 0.0)
$      radii(3) = FPD(1j+8) / radii(3)
vert(6,1) = vert(6,1) * radii(3)
vert(6,2) = vert(6,2) * radii(3)
vert(6,3) = vert(6,3) * radii(3)
vert(7,1) = vert(4,1) + vert(6,1)
vert(7,2) = vert(4,2) + vert(6,2)
vert(7,3) = vert(4,3) + vert(6,3)
if(m .eq. 1) then
    if(FPD(1k+2) .gt. vert(7,1)) goto 209
else if(m .eq. 2) then
    if(FPD(1k+4) .gt. vert(7,2)) goto 209
else if(m .eq. 3) then
    if(FPD(1k+6) .gt. vert(7,3)) goto 209
else if(m .eq. 4) then
    if(FPD(1k+2) .gt. vert(7,1)) goto 209
else if(m .eq. 5) then
    if(FPD(1k+4) .gt. vert(7,2)) goto 209
else
    if(FPD(1k+6) .gt. vert(7,3)) goto 209
endif

C      check max

$      radii(2) = radii(1) * (vert(3,1) *
$      FPD(1j+5) + vert(3,2) * FPD(1j+6)
$      + vert(3,3) * FPD(1j+7))
vert(5,1) = radii(2) * radii(1) *
$      FPD(1j+5)

```

```

vert(5,2) = radii(2) * radii(1) *
$   FPD(lj+6)
vert(5,3) = radii(2) * radii(1) *
$   FPD(lj+7)
vert(6,1) = vert(3,1) - vert(5,1)
vert(6,2) = vert(3,2) - vert(5,2)
vert(6,3) = vert(3,3) - vert(5,3)
radii(3) = sqrt(vert(6,1) *
$   vert(6,1) + vert(6,2) * vert(6,2)
$   + vert(6,3) * vert(6,3))
if(radii(3) .ne. 0)
$   radii(3) = FPD(lj+8) / radii(3)
vert(6,1) = vert(6,1) * radii(3)
vert(6,2) = vert(6,2) * radii(3)
vert(6,3) = vert(6,3) * radii(3)
vert(7,1) = vert(4,1) + vert(6,1)
vert(7,2) = vert(4,2) + vert(6,2)
vert(7,3) = vert(4,3) + vert(6,3)
if(m .eq. 1) then
    if(FPD(lk+3) .lt. vert(7,1)) goto 209
else if(m .eq. 2) then
    if(FPD(lk+5) .lt. vert(7,2)) goto 209
else if(m .eq. 3) then
    if(FPD(lk+7) .lt. vert(7,3)) goto 209
else if(m .eq. 4) then
    if(FPD(lk+3) .lt. vert(7,1)) goto 209
else if(m .eq. 5) then
    if(FPD(lk+5) .lt. vert(7,2)) goto 209
else
    if(FPD(lk+7) .lt. vert(7,3)) goto 209
endif
208 continue
    write(IOUT,(' '* / CODE ZONE ',i3,
$   ' : BODY # ',i3,' INSIDE BODY # ',
$   i3,' (RCC IN RPP)')) i,j,k
209 continue
    else if(obodyk .eq. 'SPH') then
C   Project sphere center onto axis and then find difference
C   (projected - center), scale to cylinder radius and add to
C   center of radius base. Compare distance from here to sphere
C   center against sphere radius
vert(2,1) = FPD(lk+2) - FPD(lj+2)
vert(2,2) = FPD(lk+3) - FPD(lj+3)
vert(2,3) = FPD(lk+4) - FPD(lj+4)
vert(3,1) = vert(2,1) * FPD(lj+5)
$   * radii(1)
vert(3,2) = vert(2,2) * FPD(lj+6)
$   * radii(1)
vert(3,3) = vert(2,3) * FPD(lj+7)
$   * radii(1)
vert(4,1) = vert(3,1) - vert(2,1)
vert(4,2) = vert(3,2) - vert(2,2)

```

```

vert(4,3) = vert(3,3) - vert(2,3)
radii(2) = sqrt(vert(4,1) *
$      vert(4,1) + vert(4,2) * vert(4,2)
$      + vert(4,3) * vert(4,3))
if(radii(2) .ne. 0.0)
$      radii(2) = FPD(1j+8) / radii(2)
vert(5,1) = vert(4,1) * radii(2)
$      + FPD(1j+2)
vert(5,2) = vert(4,2) * radii(2)
$      + FPD(1j+3)
vert(5,3) = vert(4,3) * radii(2)
radii(3) = sqrt((FPD(1k+2) - vert(5,1)) *
$      (FPD(1k+2) - vert(5,1)) +
$      (FPD(1k+3) - vert(5,2)) *
$      (FPD(1k+3) - vert(5,2)) +
$      (FPD(1k+4) - vert(5,3)) *
$      (FPD(1k+4) - vert(5,3)))
if(radii(3) .gt. FPD(1k+5)) goto 207
C      same with cylinder top
vert(2,1) = FPD(1k+2) - vert(1,1)
vert(2,2) = FPD(1k+3) - vert(1,2)
vert(2,3) = FPD(1k+4) - vert(1,3)
vert(3,1) = vert(2,1) * FPD(1j+5)
$      * radii(1)
vert(3,2) = vert(2,2) * FPD(1j+6)
$      * radii(1)
vert(3,3) = vert(2,3) * FPD(1j+7)
$      * radii(1)
vert(4,1) = vert(3,1) - vert(2,1)
vert(4,2) = vert(3,2) - vert(2,2)
vert(4,3) = vert(3,3) - vert(2,3)
radii(2) = sqrt(vert(4,1) *
$      vert(4,1) + vert(4,2) * vert(4,2)
$      + vert(4,3) * vert(4,3))
if(radii(2) .ne. 0.0)
$      radii(2) = FPD(1j+8) / radii(2)
vert(5,1) = vert(4,1) * radii(2)
$      + FPD(1j+2)
vert(5,2) = vert(4,2) * radii(2)
$      + FPD(1j+3)
vert(5,3) = vert(4,3) * radii(2)
radii(3) = sqrt((vert(1,1) - vert(5,1)) *
$      (vert(1,1) - vert(5,1)) +
$      (vert(1,2) - vert(5,2)) *
$      (vert(1,2) - vert(5,2)) +
$      (vert(1,3) - vert(5,3)) *
$      (vert(1,3) - vert(5,3)))
if(radii(3) .gt. FPD(1k+5)) goto 207
write(IOUT,(' '* CODE ZONE ',i3,
$      ' : BODY # ',i3,' INSIDE BODY # ',
$      i3,' (RCC IN SPH)')) i,j,k
207      continue

```

```

                                else if(obodyk .eq. 'RCC') then
C      get length of outer axis for scaling purposes
                                radii(2) = 1.0 / sqrt(FPD(1k+5) *
$                                FPD(1k+5) + FPD(1k+6) * FPD(1k+6) +
$                                FPD(1k+7) * FPD(1k+7))
C      see if bottom of inner cylinder pokes through side of outer
C      project center of inner bottom onto outer axis then subtract
C      to get vector from outer axis to inner bottom center. Project
C      this onto inner axis and subtract and scale to get point on
C      inner bottom farthest from outer axis. Compare distance to axis
C      to outer radius... then repeat for top
                                vert(2,1) = FPD(1j+2) - FPD(1k+2)
                                vert(2,2) = FPD(1j+3) - FPD(1k+3)
                                vert(2,3) = FPD(1j+4) - FPD(1k+4)
                                radii(3) = vert(2,1) * FPD(1k+5) +
$                                vert(2,2) * FPD(1k+6) +
$                                vert(2,3) * FPD(1k+7)
                                if(radii(3) .lt. 0.0) goto 211
                                vert(3,1) = radii(3) * radii(2) *
$                                radii(2) * FPD(1k+5)
                                vert(3,2) = radii(3) * radii(2) *
$                                radii(2) * FPD(1k+6)
                                vert(3,3) = radii(3) * radii(2) *
$                                radii(2) * FPD(1k+7)
C      vert(4,?) is vector from outer axis to inner bottom center
                                vert(4,1) = vert(2,1) - vert(3,1)
                                vert(4,2) = vert(2,2) - vert(3,2)
                                vert(4,3) = vert(2,3) - vert(3,3)
                                radii(3) = vert(4,1) * FPD(1j+5) +
$                                vert(4,2) * FPD(1j+6) +
$                                vert(4,3) * FPD(1j+7)
                                vert(3,1) = radii(3) * radii(1) *
$                                radii(1) * FPD(1j+5)
                                vert(3,2) = radii(3) * radii(1) *
$                                radii(1) * FPD(1j+6)
                                vert(3,3) = radii(3) * radii(1) *
$                                radii(1) * FPD(1j+7)
                                if(radii(3) .eq. 0.0) then
                                    vert(3,1) = vert(4,1)
                                    vert(3,2) = vert(4,2)
                                    vert(3,3) = vert(4,3)
                                endif
C      vert(5,?) is vector in direction of point on bottom furthest
C      from the outer axis
                                vert(5,1) = vert(4,1) - vert(3,1)
                                vert(5,2) = vert(4,2) - vert(3,2)
                                vert(5,3) = vert(4,3) - vert(3,3)
                                radii(4) = sqrt(vert(5,1) *
$                                vert(5,1) + vert(5,2) * vert(5,2) +
$                                vert(5,3) * vert(5,3))
C      test for coaxial case at this point
                                if(radii(4) .ne. 0.0)

```

```

$                                radii(4) = FPD(1j+8) / radii(4)
C    vert(6,?) is the point on the bottom furthest from the axis
$                                vert(6,1) = FPD(1j+2) + vert(5,1) *
$                                radii(4)
$                                vert(6,2) = FPD(1j+3) + vert(5,2) *
$                                radii(4)
$                                vert(6,3) = FPD(1j+4) + vert(5,3) *
$                                radii(4)
C    find distance between point and axis and compare to radius
$                                vert(10,1) = vert(6,1) - FPD(1k+2)
$                                vert(10,2) = vert(6,2) - FPD(1k+3)
$                                vert(10,3) = vert(6,3) - FPD(1k+4)
$                                vert(11,1) = vert(10,2) * FPD(1k+7) -
$                                vert(10,3) * FPD(1k+6)
$                                vert(11,2) = vert(10,3) * FPD(1k+5) -
$                                vert(10,1) * FPD(1k+7)
$                                vert(11,3) = vert(10,1) * FPD(1k+6) -
$                                vert(10,2) * FPD(1k+5)
$                                radii(5) = sqrt(vert(10,1) * vert(10,1)
$                                + vert(10,2) * vert(10,2)
$                                + vert(10,3) * vert(10,3))
$                                if(radii(5) * radii(2) .gt. FPD(1k+8))
$                                goto 211
C    now do the same for the top
$                                vert(2,1) = vert(1,1) - FPD(1k+2)
$                                vert(2,2) = vert(1,1) - FPD(1k+3)
$                                vert(2,3) = vert(1,1) - FPD(1k+4)
$                                radii(3) = vert(2,1) * FPD(1k+5) +
$                                vert(2,2) * FPD(1k+6) +
$                                vert(2,3) * FPD(1k+7)
$                                if(radii(3) .lt. 0.0) goto 211
$                                vert(3,1) = radii(3) * radii(2) *
$                                radii(2) * FPD(1k+5)
$                                vert(3,2) = radii(3) * radii(2) *
$                                radii(2) * FPD(1k+6)
$                                vert(3,3) = radii(3) * radii(2) *
$                                radii(2) * FPD(1k+7)
C    vert(4,?) is vector from outer axis to inner bottom center
$                                vert(4,1) = vert(2,1) - vert(3,1)
$                                vert(4,2) = vert(2,2) - vert(3,2)
$                                vert(4,3) = vert(2,3) - vert(3,3)
$                                radii(3) = vert(4,1) * FPD(1j+5) +
$                                vert(4,2) * FPD(1j+6) +
$                                vert(4,3) * FPD(1j+7)
$                                vert(3,1) = radii(3) * radii(1) *
$                                radii(1) * FPD(1j+5)
$                                vert(3,2) = radii(3) * radii(1) *
$                                radii(1) * FPD(1j+6)
$                                vert(3,3) = radii(3) * radii(1) *
$                                radii(1) * FPD(1j+7)
$                                if(radii(3) .eq. 0.0) then
$                                vert(3,1) = vert(4,1)

```

```

                                vert(3,2) = vert(4,2)
                                vert(3,3) = vert(4,4)
                                endif
C    vert(5,?) is vector in direction of point on bottom furthest
C    from the outer axis
                                vert(5,1) = vert(4,1) - vert(3,1)
                                vert(5,2) = vert(4,2) - vert(3,2)
                                vert(5,3) = vert(4,3) - vert(3,4)
                                radii(4) = sqrt(vert(5,1) *
$                                vert(5,1) + vert(5,2) * vert(5,2) +
$                                vert(5,3) * vert(5,3))
                                if(radii(4) .ne. 0.0)
$                                radii(4) = FPD(1j+8) / radii(4)
C    vert(6,?) is the point on the bottom furthest from the axis
                                vert(6,1) = vert(1,1) + vert(5,1) *
$                                radii(4)
                                vert(6,2) = vert(1,1) + vert(5,2) *
$                                radii(4)
                                vert(6,3) = vert(1,1) + vert(5,3) *
$                                radii(4)
C    find distance between point and axis and compare to radius
                                vert(10,1) = vert(6,1) - FPD(1k+2)
                                vert(10,2) = vert(6,2) - FPD(1k+3)
                                vert(10,3) = vert(6,3) - FPD(1k+4)
                                vert(11,1) = vert(10,2) * FPD(1k+7) -
$                                vert(10,3) * FPD(1k+6)
                                vert(11,2) = vert(10,3) * FPD(1k+5) -
$                                vert(10,1) * FPD(1k+7)
                                vert(11,3) = vert(10,1) * FPD(1k+6) -
$                                vert(10,2) * FPD(1k+5)
                                radii(5) = sqrt(vert(10,1) * vert(10,1)
$                                + vert(10,2) * vert(10,2)
$                                + vert(10,3) * vert(10,3))
                                if(radii(5) * radii(2) .gt. FPD(1k+8))
$                                goto 211
C    now check inner cylinder against outer cylinder ends
212                                radii(3) = (FPD(1k+5) * FPD(1j+5) +
$                                FPD(1k+6) * FPD(1j+6) +
$                                FPD(1k+7) * FPD(1j+7)) *
$                                radii(1) * radii(1)
                                vert(2,1) = FPD(1j+5) * radii(3)
                                vert(2,2) = FPD(1j+6) * radii(3)
                                vert(2,3) = FPD(1j+7) * radii(3)
                                vert(3,1) = FPD(1k+5) - vert(2,1)
                                vert(3,2) = FPD(1k+6) - vert(2,2)
                                vert(3,3) = FPD(1k+7) - vert(2,3)
                                radii(4) = sqrt(vert(3,1) * vert(3,1) +
$                                vert(3,2) * vert(3,2) +
$                                vert(3,3) * vert(3,3))
C    check for same directionness in axes
                                if(radii(4) .eq. 0.0) then
C    check inner top and bottom center...

```

```

C    project them onto outer axis relative to outer bottom
      vert(4,1) = FPD(1j+2) - FPD(1k+2)
      vert(4,2) = FPD(1j+3) - FPD(1k+3)
      vert(4,3) = FPD(1j+4) - FPD(1k+4)
      radii(5) = (FPD(1k+5) * vert(4,1) +
$          FPD(1k+6) * vert(4,2) +
$          FPD(1k+7) * vert(4,3)) * radii(2)
      vert(5,1) = vert(1,1) - FPD(1k+2)
      vert(5,2) = vert(1,2) - FPD(1k+3)
      vert(5,3) = vert(1,3) - FPD(1k+4)
      radii(6) = (FPD(1k+5) * vert(5,1) +
$          FPD(1k+6) * vert(5,2) +
$          FPD(1k+7) * vert(5,3)) * radii(2)
C    compare project length to outer axis length
      if((radii(5) .lt. 0.0) .or.
$          (radii(5) .gt. radii(2)) .or.
$          (radii(6) .lt. 0.0) .or.
$          (radii(6) .gt. radii(2)))
$          goto 211
      goto 213
C    do not continue out of this if block
      endif
C    scale length to inner radius
      radii(4) = FPD(1j+8) / radii(4)
      vert(3,1) = vert(3,1) * radii(4)
      vert(3,2) = vert(3,2) * radii(4)
      vert(3,3) = vert(3,3) * radii(4)
C    add and subtract from inner bottom and top center
      vert(4,1) = FPD(1j+2) + vert(3,1)
      vert(4,2) = FPD(1j+3) + vert(3,2)
      vert(4,3) = FPD(1j+4) + vert(3,3)
      vert(5,1) = FPD(1j+2) - vert(3,1)
      vert(5,2) = FPD(1j+3) - vert(3,2)
      vert(5,3) = FPD(1j+4) - vert(3,3)
      vert(6,1) = vert(1,1) + vert(3,1)
      vert(6,2) = vert(1,2) + vert(3,2)
      vert(6,3) = vert(1,3) + vert(3,3)
      vert(7,1) = vert(1,1) - vert(3,1)
      vert(7,2) = vert(1,2) - vert(3,2)
      vert(7,3) = vert(1,3) - vert(3,3)
C    project onto outer axis relative to outer bottom center
      do 229 m=4,7
      vert(8,1) = vert(m,1) - FPD(1k+2)
      vert(8,2) = vert(m,2) - FPD(1k+3)
      vert(8,3) = vert(m,3) - FPD(1k+4)
      radii(5) = ((vert(8,1) * FPD(1k+5)) +
$          (vert(8,2) * FPD(1k+6)) +
$          (vert(8,3) * FPD(1k+7))) * radii(2)
C    compare lengths to length of outer axis
      if((radii(5) .lt. 0.0) .or.
$          (radii(5) .gt. FPD(1k+8)))
$          goto 211

```

```

229                                continue
C   by golly, it is inside!
213                                write(IOUT, '('*/ CODE ZONE ',i3,
$                                ' : BODY # ',i3,' INSIDE BODY # ',
$                                i3,' (RCC IN RCC)')') i,j,k
211                                continue
                                endif
                                endif
                                endif
                                endif
193                                continue
192                                continue
191                                continue

```

VITA

Reuben Dale Hochstedler was born on January 10, 1973 in Kokomo, Indiana to Rachael Marie and Robert Ivan Hochstedler. During his elementary school years, he was advanced from the second directly to the fourth grade within a group of other talented students. In May of 1990, he graduated summa cum laude from Eastern High School and was one of his class's three valedictorians.

In July of 1990, Reuben began his undergraduate studies at the Rose-Hulman Institute of Technology as a member of its Fast Track Calculus program. While at Rose, he became a member of the Alpha Phi Omega, National Service Fraternity in which he has been multiply awarded and remained actively involved. In May of 1994, he graduated magna cum laude with a Bachelor of Science in Electrical Engineering and a minor in Computer Science.

Reuben accepted a Graduate Research Assistantship at the University of Tennessee Space Institute where he completed the requirements for his Master of Science degree for a December 1996 graduation. He has also accepted a Graduate Student Instructorship at the University of Michigan where he will begin work towards his doctorate in Computer Science and Engineering in September of 1996.