The Monte Carlo Method: Versatility Unbounded In A Dynamic Computing World
Chattanooga, Tennessee, April 17–21, 2005, on CD-ROM, American Nuclear Society, LaGrange Park, IL (2005)

APPLICATION OF A JAVA-BASED, UNIVEL GEOMETRY, NEUTRAL PARTICLE MONTE CARLO CODE TO THE SEARCHLIGHT PROBLEM

Joshua J. Cogliati and Charles A. Wemple
Idaho National Laboratory
P.O. Box 1625
Idaho Falls, ID 83415-3885 USA
cogljj@inel.gov, cew@inel.gov

ABSTRACT

A univel geometry, neutral particle Monte Carlo transport code, written entirely in the Java programming language, is under development for medical radiotherapy applications. The code uses ENDF-VI based continuous energy cross section data in a flexible XML format. Full neutron-photon coupling, including detailed photon production and photonuclear reactions, is included. Charged particle equilibrium is assumed within the patient model so that detailed transport of electrons produced by photon interactions may be neglected. External beam and internal distributed source descriptions for mixed neutron-photon sources are allowed. Flux and dose tallies are performed on a univel basis. A four-tap, shift-register-sequence random number generator is used.

Initial verification and validation testing of the basic neutron transport routines is underway. The searchlight problem was chosen as a suitable first application because of the simplicity of the physical model. Results show excellent agreement with analytic solutions. Computation times for similar numbers of histories are within a factor of two of other neutron MC codes written in C and FORTRAN.

Key Words: Monte Carlo, neutron transport, Java, univel, searchlight problem

1 INTRODUCTION

The history of random sampling techniques in scientific analysis has roots which predate automated (both mechanical and electronic) computing devices. However, the modern application of random and statistical sampling methods, and, thus, the Monte Carlo method for particle transport, is intimately tied to the development and evolution of the electronic computer (Chapter 2 of the MCNP [1] manual has an excellent condensed history of this development). The computer has evolved considerably since the early days of ENIAC, and a plethora of Monte Carlo codes for various purposes have been developed (MCNP [1], VIM [2], MORSE [3], COG [4], HETC [5], EGSnrc [6], RAFFLE [7], etc.), but one aspect of Monte Carlo codes has remained fixed since the 1960s – they are all written primarily using the Fortran programming language.

While Fortran and similar scientific programming languages are well suited for intensive computation, they suffer from a lack of easy portability between machine architectures, operating systems, and compilers. It has been only recently that advances in software languages, such as object-oriented programming, have been adapted to the Monte Carlo method. For example, the Geant4 [8] code uses object-oriented programming techniques in the C++ language. However, while these applications take advantage of the enhanced maintainability and modular design of object-oriented programming, they still suffer from portability problems and a dependency on external toolkits for user interface design.

The advent of the Java programming language [9] presented a solution to these shortcomings. Its concept of "compile once, run everywhere" virtually eliminated portability difficulties and the incorporation of a full native windowing toolkit allowed integrated development of user interfaces with computational applications. Unfortunately, early testing of compute-intensive applications written in Java showed significant penalties in runtime compared to optimized Fortran and turned most scientific programmers away from Java. However, significant improvements have been made in the language and compiler designs to better accommodate computationally demanding applications.

These improvements in the capabilities of Java suggested that it could be suitable for development of large-scale, computationally intensive, scientific applications. In October 2001, the development of the Minerva radiation treatment planning system [10-12] was initiated as a joint effort of the Idaho National Engineering and Environmental Laboratory, Montana State University, and Lawrence Livermore National Laboratory. The goal of the Minerva project was to develop a patient-centric, multi-modal, radiation treatment planning system written entirely in the Java programming language.

One aspect of this project is development of a coupled neutron/photon transport code for external beam neutron radiotherapy and neutron capture therapy (NCT). This code, written entirely in Java, uses the univel geometry method [13] for rapid particle tracking, an XML-based cross section format, an extended period random number generator, and univel-based tallies. It also exploits many of the advantages provided by the Java language and object oriented coding principles.

The remainder of this paper provides details of the development of the JART (JAva Radiation Transport) code. The coding, data, and mathematical methods employed are described in Section 2. The first application of the code, to the searchlight problem (a classic radiation transport benchmark case), is presented in Section 3. Finally, the plans for future development are discussed in Section 4.

2 METHODS

2.1 Java and Object Oriented Coding

Java is an object oriented programming language. An object is similar to a Fortran 90 derived type or a C structure, except it has additional features. Like derived types, an object has components of simpler types, called fields. However, unlike derived types, objects also have functions associated with them, called methods.

This leads to the first ability of objects: inheritance. One object can inherit from another object. When this happens, the new object has all the original fields and methods of the object that it inherits from, but it can also add new fields and new methods. In addition, methods can be defined to override inherited methods of the same name. The new object can then be used exactly as the old object could be, but can also be used to take advantage of the expanded capabilities provided by the new methods and data.

A second ability of object oriented Java is interfaces. An interface is a list of methods. An object can implement an interface by having defining all the methods in the interface's list. Then, code can be written to use an interface without knowing how the methods are defined or where

they are defined. In this fashion, a function could use an interface by using some of the methods in the interface, all with a new object that was written after the original function.

Another advantage of Java is its large built in library. Like the Fortran and C libraries, the Java library contains mathematical functions, such as sin and sqrt. However, Java's library also contains networking, database access, graphical user interface (GUI) code, and more. Since this library is available on each platform that Java runs on, code that uses this library can run unmodified on each of those platforms. This makes Java substantially more portable.

Finally, the overall portability and maintainability of the code is enhanced by the "write once, run everywhere" philosophy of Java. Java code can be compiled to class files that run on the Java Virtual Machine (JVM). There are implementations of the JVM on multiple operating systems and hardware platforms that allow the class files to run on these platforms. This provides a great deal of flexibility for the code developer, as peculiarities of different operating systems and compilers no longer must be included in the code. As a side effect, this eliminates the "ifdef" construct and other similar methods for including these dependencies and greatly simplifies code maintenance.

2.2 Uniform Volume Element (Univel) Tracking

The solution space defined in the input is subdivided into univels, which are rectangular parallelepipeds of uniform size. The transport method uses a three-dimensional variant [13,17] of Bresenham's line drawing algorithm [18] for the ray tracing and particle tracking. Enhancements to the algorithm were made to incorporate an improved method for calculating the total distance traveled across each univel, which is a necessary quantity for calculating tallies on a univel basis. A summary of the revised tracking algorithm is presented below.

The univel tracking algorithm is initialized with a location and a direction. This initialization occurs at the birth of a particle and at each collision. The distance to collision is sampled in the normal fashion. At each univel boundary crossing, a check is performed to see if a new material is encountered; if so, a new distance to collision is sampled, otherwise the original distance to collision minus the distance in the univel is used. Because of the uniform nature of the univels, the distance traversed between boundary crossings along each spatial coordinate axis is a constant and can be calculated from the initialization values and the univel dimensions. This allows the use of primarily integer arithmetic to perform the particle tracking through the univels. Calculating the distance traveled in a univel uses one floating point subtraction and one addition per univel crossing. Univel material properties can be stored in an array and accessed quickly.

The tally calculation uses a basic track length estimator, which requires the distance traveled through each univel. This is calculated at the same time as the univel tracking program by the following method. As noted above, the same distance is traveled between each parallel boundary crossing along a given line. For example, each time a boundary is crossed along the x-direction, the distance that has been traveled since the previous boundary crossing in this direction can be calculated. The distance traveled in each univel is determined by knowing that each time a particle crosses a univel boundary, it has traveled a known, constant distance. This distance and a total distance are stored for each dimension, along with an overall total distance. The overall total distance is the distance at the last boundary crossing and is updated at each boundary crossing.

A two-dimensional example of this procedure is shown in Figure 1. The incremental distance in the x-direction is denoted as Δx and as Δy in the y-direction. For the following example, let the total distance traveled at each x-direction boundary crossing be denoted as X and the total distance traveled at each y-direction crossing as Y. The total distance traveled can then be calculated by adding the incremental distance traveled to the last total distance for that dimension. Thus, the

total distance traveled at point B is Δx added to the total distance X traveled at point A. When the particle reaches point C, the distance Δy is added to the last Y total distance. When point D is reached, Δx is added to the last calculated value of X, which was the total distance at point B. The same process occurs for each of the directions. The distance traveled in each univel is the difference between the total distance for the dimensional boundary crossed and the overall total distance. The particle tracking ceases when the weight reaches a user-defined cutoff or the edge of the univel space is reached.

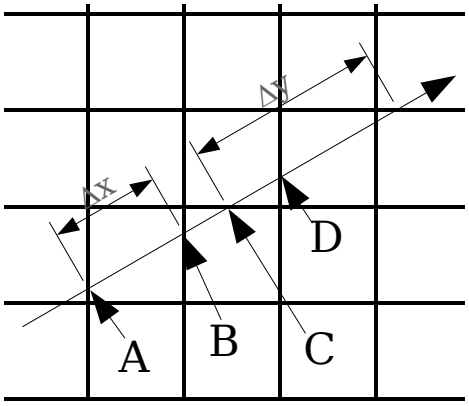

Figure 1. A two-dimensional example of ray tracing through the univel geometry.

2.3 Tallies and Statistics

The tallies are calculated on the univel grid defined by the problem geometry or an integer multiple of this univel grid. A standard track length estimator is used for the tallies, i.e., each time a particle passes through a univel, the distance that the particle travels times the weight of that particle is added to the tally in the univel. Tally values at a point are calculated by quadratic interpolation along the three coordinate axes, asssuming that the value for a univel applies at the centroid of the univel volume. This interpolation uses the tally values for the univel containing the point and the 26 surrounding univels in 3-space. Response functions are folded with the track length estimator at each tally to calculate the dose components.

The tally statistics estimate the deviation of the calculated tally results from the "true" results, those derived from an infinitely long calculation. It should be noted that this is not necessarily how close the answer is to the correct answer, since there are biases in the data and methods used. There are two extreme ways of calculating the "true" result. The first is to treat each particle tracked as a single experiment. The second is to treat the entire run of particles as one experiment. The first method requires statistics computation for each particle that is tracked. When this was done, the transport code was spending far more time calculating the statistics than performing the particle tracking. The second extreme does not give any idea about the precision of the calculation, since the variance of a single experiment cannot be calculated. The method used in JART allows the user to define the balance between the frequency of statistical value computation and overall calculation speed.

The variance of the aggregated experiments is expected to be less than the variance of an individual experiment, since the aggregation will smooth the data, and the overall variance of the estimate is expected to be comparable. To prove this, it must be shown that the mean and the variance of the mean are the same regardless of how the experiments are aggregated.

The number of experiments that are aggregated is referred to here as the stride, r. In practice, the total number of particles, n, is required to be an integer multiple of the stride. The mean and variance of the mean for single particle experiments (r=1) are given by:

$$\bar{x} = \sum_{i=1}^{n} x_i$$

$$\mathrm{var}(\bar{x}) = \frac{\mathrm{var}(x_i)}{n}$$

The definition of an aggregated experiment with the same total number of particle histories is:

$$\bar{y}_j = \frac{\sum_{i=1}^{r} x_{i+jr}}{r}$$

and the mean of the aggregated experiments is:

$$\bar{x}_{agg} = \frac{\sum_{j=1}^{\frac{n}{r}} \bar{y}_j}{\frac{n}{r}} = \frac{r\sum_{j=1}^{\frac{n}{r}} \bar{y}_j}{n}$$

Substituting the definition of the aggregated mean gives:

$$\frac{\sum_{j=1}^{\frac{n}{r}} \sum_{i=1}^{r} x_{i+jr}}{n} = \frac{\sum_{i=1}^{n} x_i}{n} = \bar{x}$$

Therefore, the mean value will be the same for any stride.

Next, the variance of the aggregated mean needs to be examined. This is defined as:

$$\mathrm{var}(\bar{y}_j) = \mathrm{var}\left(\frac{\sum_{i=1}^{r} x_{i+jr}}{r}\right) = \frac{1}{r^2}\sum_{i=1}^{r} \mathrm{var}(x_{i+jr}) = \frac{1}{r^2} r\,\mathrm{var}(x_i) = \frac{\mathrm{var}(x_i)}{r}$$

The variance of the $\bar{y}_j$ will decrease as the stride increases. For statistical error calculation the variance of the the mean, $\bar{x}$ needs to be found. As the following shows, this can be determined from the variance of the $\bar{y}_j$:

$$\mathrm{var}(\bar{x}_{agg}) = \frac{\mathrm{var}(\bar{y}_j)}{\frac{n}{r}} = \frac{\frac{\mathrm{var}(x_i)}{r}}{\frac{n}{r}} = \frac{\mathrm{var}(x_i)}{n} = \mathrm{var}(\bar{x})$$

The tally calculation is performed every r particle histories. The mean of the aggregated experiments is calculated, which is used in combination with other aggregated means and the sample size to calculate the mean and the variance of the mean.

2.4 Other Issues

Continuous energy cross section data are used in the JART code. The format is an extensible markup language (XML) file based on the MCNP ACE data format [1]. The new XML file is fully reaction ordered, with all data pertaining to a particular reaction (cross section, secondary particle

energy and angular distributions, etc.) grouped together. This provides a compact format description and a far more human readable data set. In addition, nuclide data sets contain all relevant data types – neutron, photoatomic, photonuclear, and S(α,β) – in one file. This is possible because there is no need for multiple temperature data sets in medical radiotherapy and the flexibility of the MCNP "mix-and-match" format is not necessary.

A four-tap, shift-register-sequence random number generator [14] is used. This provides a mathematically robust, long period generator with greatly reduced correlation. The generator for the n^{th} random number is given by:

$$r[(n\&M)] = r[(n-A)\&M] \oplus r[(n-B)\&M] \oplus r[(n-C)\&M] \oplus r[(n-D)\&M]$$

where r is an array of length M, and the constants are A=471, B=1586, C=6988, D=9689, and M=16383. In this equation, & denotes the bitwise "and" operation and $\oplus$ denotes the bitwise "exclusive-or" operation. Note that the generator is constantly updating the array r as each new random number is calculated. The "and" with M in the generator essentially loops endlessly through the list if M+1 is a power of 2. Only 64 kB of memory is required if 32-bit integers are used. Division by the maximum integer value gives a random number on [0,1]. The period of the RNG is given by 2^p-1, where p is the maximum tap value; in this case, the period is 2^{9689}-1.

The maximum size of the geometry is mainly limited by memory. Each univel requires two bytes to store the material number (allowing 2^{16} distinct materials). Additional memory will be used depending on the tallies that are requested, with each tally type requiring 8 bytes per univel cell.

3 APPLICATION TO SEARCHLIGHT PROBLEM

3.1 Description of Problem

The searchlight problem is a classical two-dimensional radiation transport problem originally proposed by Chandrasekhar [15]. The problem as considered for this work consists of a homogeneous half-space of material illuminated by a pencil beam of monoenergetic neutrons normal to the free surface. The solution assumes no energy dependence of the cross section and only isotropic elastic scattering. The composition of the half-space is a material with unit total cross section which is 99% scattering and 1% absorbing. This provides an excellent problem for testing the basic mechanics of the Monte Carlo transport, especially the tracking and tally routines.

The reference solution for this work is the analytical solution derived by Ganapol [16]. Computations were performed for the radial dependence of the flux at several penetration depths within the half-space. MCNP calculations to simulate this analytical solution were also performed [16] using version 4C of the code. A brief description of these calculations follows for clarity. The half-space was modeled as a 30cm radius hemisphere. The flux tallies used annular shells for surface (type 2) tallies. A special monoenergetic cross section library was created which contained only two "isotopes" - one pure elastic scatterer with infinite (300 amu) mass; the other a pure absorber.

The JART model approximated the half-space by a rectangular parallelepiped of size 36x36x38 mean free paths (mfp), which adequately represents an infinite half-space. The univel size used for the calculation was 0.1x0.1x0.1 mfp. The flux as a function of radius was calculated

at several different penetration depths. The JART tallies at each radial point were averaged over the four points in the primary compass directions to exploit the natural symmetry of the problem. The JART calculation used the same cross section library as the MCNP calculations.

Two additional calculations were performed using the SeraMC code [19, 20], which was designed specifically for radiation transport in neutron therapy applications. The SeraMC code uses the same physics as JART, except that SeraMC uses multigroup cross section data and is coded in FORTRAN with some C code. The geometry model used in one of the SeraMC calculations is similar to the MCNP4C geometry description (combinatorial geometry, or CG) and the other uses a univel geometry model similar to the JART calculation.

3.2 Results

The results of the JART calculations, along with the reference solution and an equivalent MCNP calculation, are shown in Figure 2. The tally statistics on the JART and MCNP results are too small to be resolved on the images. Agreement between the various methods is excellent and within the calculated tally statistics of the Monte Carlo results. This provides verification for the correct functioning of the basic transport mechanics of the JART code.

The results displayed in Figure 2 required simulation of 32×10^6 particle histories. The CPU time consumed by the JART calculation was 291 minutes, compared with 161 minutes for the MCNP calculation, 297 minutes for the CG geometry SeraMC, and 533 minutes for the univel SeraMC. All simulations were run on a single processor Sun Blade 2000 workstation. The ratio of the computation times of JART and MCNP, which is less than a factor of two, shows that Java has the capability to perform complex scientific calculations without an extreme penalty in computation time. Because of the nature of the univel geometry, increased complexity in the problem geometry, as may be expected for models built from human medical images, will have minimal effect on the JART runtimes but can greatly increase the runtime for MCNP. Other work has shown that SeraMC executes faster than MCNP for realistic radiotherapy applications [21,22]. It is possible that JART could have an advantage in computation time over MCNP for clinical radiotherapy calculations. This will be verified in future investigations.

The testing also included verification of the revised computation methods of the tally statistics in JART. In order to verify that the statistics stride size has no effect on the standard error calculations for JART, a a series of JART runs with varying statistical stride were performed. Four radial tally locations – 0.15cm, 0.55cm, 0.95cm, and 2.25cm – at a depth of 1.25cm were selected for this calculation. The results are displayed in Table I, and show essentially no variation in the standard error with statistical stride size.

4 CONCLUSIONS

A Java-based Monte Carlo transport code using the univel geometry has been described. The results of initial testing benchmark calculations have been presented and show that the Java-based code, JART, reproduces the searchlight problem with runtimes within a factor of two of existing C- and Fortran-based Monte Carlo codes. When considered with the extensive capabilities of Java and the relative infancy of Java compiler technology, this demonstrates that the Java programming language can be appropriate for compute-intensive scientific applications.

The JART code is still a work-in-progress. Although the entire logic path for the Monte Carlo tracking and reaction simulation is in place, the actual coding for the algorithms for some of these

features remains to be completed. Remaining development tasks include completion of the scattering reaction capabilities, addition of the photon transport capabilities, improvement of the user interface and output edits, and full integration into the Minerva system. Subsequent testing, verification, and validation will also be required before JART is ready for application in clinical settings.

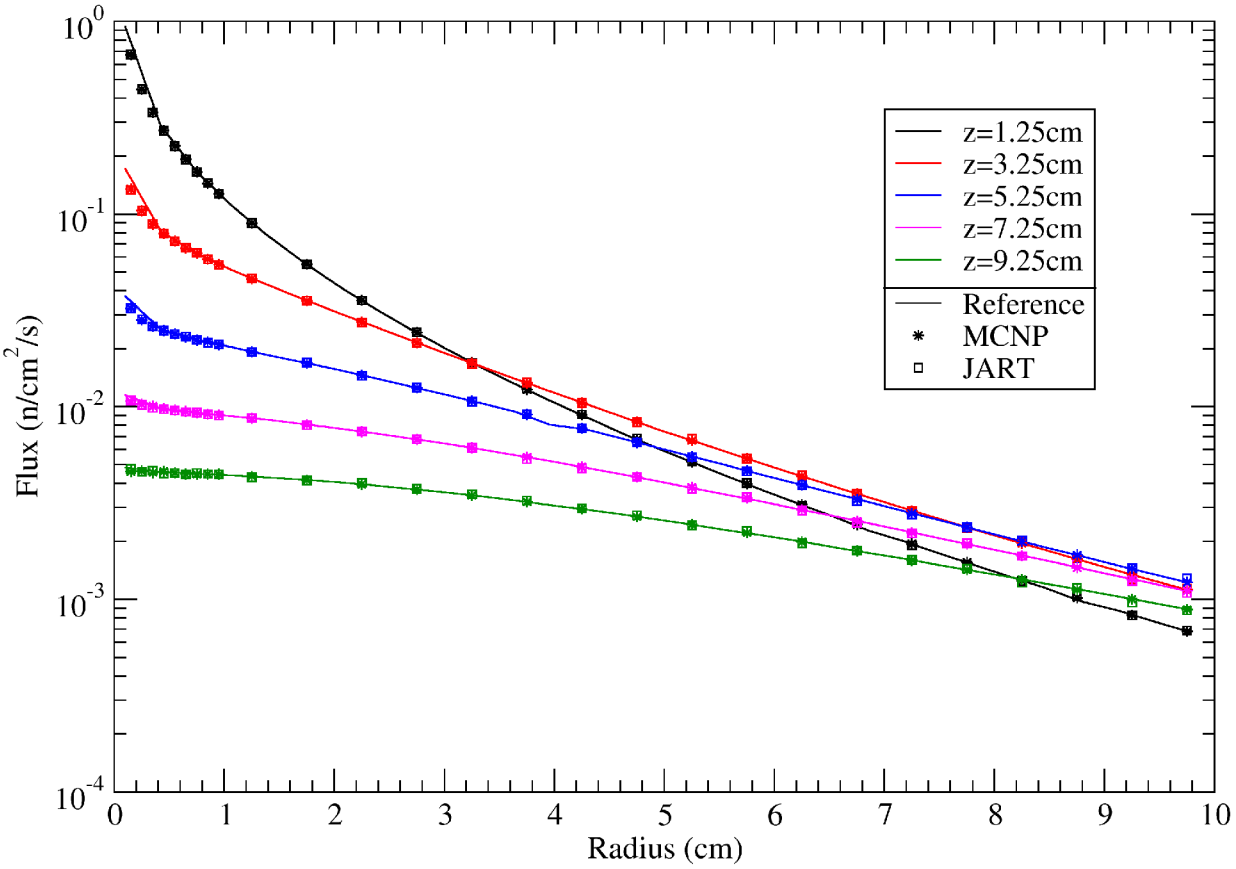

Figure 2. Comparison of searchlight problem results for several penetration depths.

Table I. Variation of JART absolute standard error with stride for selected tally points at z=1.25cm. All cases run for 3.2×10^7 histories.

	JART				
Radius (cm)	**10000**	**1000**	**100**	**10**	**1**
0.15	5.955E-4	5.976E-4	5.967E-4	5.968E-4	5.968E-4
0.55	3.557E-4	3.566E-4	3.575E-4	3.575E-4	3.575E-4
0.95	2.695E-4	2.702E-4	2.704E-4	2.705E-4	2.705E-4
2.25	1.433E-4	1.440E-4	1.440E-4	1.440E-4	1.440E-4

5 ACKNOWLEDGMENTS

This work was sponsored by the United States Department of Energy, Office of Science, under DOE Idaho Operations Office Contract DE-AC07-99ID13727.

6 REFERENCES

1. J.F. Briesmeister (ed.), "MCNP – A General Monte Carlo N-Particle Transport Code, Version 4C," LA-13709-M, Los Alamos National Laboratory (2000).
2. R.N. Blomquist, "VIM Monte Carlo Neutron/Photon Transport Code User's Guide Version 4.0," CCC-658, Radiation Safety Information Computational Center (2000).
3. M.B. Emmet, "MORSE-CGA: A Monte Carlo Radiation Transport Code with Array Geometry Capability," ORNL-6174, Oak Ridge National Laboratory (1985).
4. T.P. Wilcox, Jr., and E.M. Lent, "COG – A Particle Transport Code Designed to Solve the Boltzmann Equation for Deep-penetration (Shielding) Problems," LLNL M-221-1 (1989).
5. K.C. Chandler and T.W. Armstrong, "HETC High-Energy Nucleon-Meson Transport Code," ORNL-4744, Oak Ridge National Laboratory (1972).
6. I. Kawrakow and D.W.O. Rogers, "The EGSnrc Code System: Monte Carlo Simulation of Electron and Photon Transport," PIRS-701, National Research Council of Canada (2002).
7. F.J. Wheeler, S.A. Easson, R.A. Grimesey and D.E. Wessol, "The RAFFLE-V General Purpose Monte Carlo Code for Neutron and Gamma Transport," EGG-PHYS-6003, Idaho National Engineering Laboratory (1983).
8. S. Agostinelli, et al., "Geant4 – A Simulation Toolkit", *Nuclear Instruments and Methods A*, **506**, pp. 250-303 (2003).
9. J. Gosling, B. Joy, and G. Steele, *The Java Language Specification*, Addison-Wesley, Reading, MA, USA (1996).
10. C.A. Wemple, et al., "MINERVA – a multi-modal radiation treatment planning system," *Advanced Radiation and Isotopes*, **61**, pp. 745-752 (2004).
11. C.A. Wemple, et al., "MINERVA: A Multimodality Plugin-based Radiation Treatment Planning Software System," *Radiation Protection Dosimetry*, accepted for publication (2005).
12. J. Lehmann, et al., "Monte Carlo Treatment Planning for Targeted Radiotherapy within the MINERVA System," *Physics in Medicine and Biology*, **50**(5), pp. 947-958 (2005).
13. M.W. Frandsen, D.E. Wessol, and F.J. Wheeler, "Rapid Geometry Interrogation for a Uniform Volume Element-Based Monte Carlo Particle Transport Simulation," P*roceedings of the ANS Radiation Protection and Shielding Division Topical Meeting*, Nashville, TN, USA, April 19-23, 1998, Vol. 2, pp. 89-94 (1998).
14. R.M. Ziff, "Four-tap shift-register-sequence random-number generators," *Computers in Physics*, **12**(4) pp. 385-392 (1998).
15. S. Chandrasekhar, "On the Diffuse Reflection of a Pencil of Radiation by a Plane-Parallel Atmosphere," *Proceedings of the National Academy of Sciences*, **44**, pp.933-940 (1958).
16. B.D. Ganapol, et al., "The Searchlight Problem for Neutrons in a Semi-Infinite Medium," *Nuclear Science and Engineering*, **118**, pp. 38-53 (1994).

17. M.W. Frandsen, “Rapid Geometry Interrogation for a Uniform Volume Element-based Monte Carlo Particle Transport Simulation,” M.S. Thesis, Montana State University (1998).
18. J.E. Bresenham, "Algorithm for Computer Control of a Digital Plotter," *IBM System Journal*, **4**, pp. 25-30 (1965).
19. D.W. Nigg, C.A. Wemple, D.E. Wessol, and F.J. Wheeler, “SERA – An Advanced Treatment Planning System for Neutron Therapy and BNCT,” *ANS Transactions*, **80**, pp. 66-68 (1999).
20. D.E. Wessol, et al., “SERA: Simulation Environment for Radiotherapy Applications User's Manual Version 1C0,” INEEL/EXT-02-00698, Idaho National Engineering and Environmental Laboratory (2002).
21. J.R. Albritton and W.S. Kiger, III, “Intercomparison of Neutron Capture Therapy Treatment Planning Systems,” presented at the 11th International Congress on Neutron Capture Therapy, http://isnct.org/ISNCT-11/program-papers.html.
22. J.R. Albritton, “Analysis of the SERA Treatment Planning System and Its Use in Boron Neutron Capture Synovectomy,” Masters thesis, Massachusetts Institute of Technology, 2001.