ICANS-XIV
14th Meeting of the International Collaboration on
Advanced Neutron Sources
June 14–19, 1998
Starved Rock Lodge, Utica, Illinois, U.S.A.

THE NEUTRON INSTRUMENT MONTE CARLO LIBRARY MCLIB: RECENT DEVELOPMENTS

Philip A. Seeger, Luke L. Daemen, Rex P. Hjelm, Jr., and Thierry G. Thelliez

Manuel Lujan Jr. Neutron Scattering Center, Los Alamos National Laboratory
Los Alamos, NM 87545, U.S.A.
E-mail: PASeeger@aol.com

ABSTRACT

A brief review is given of the developments since the ICANS-XIII meeting made in the neutron instrument design codes using the Monte Carlo library MCLIB. Much of the effort has been to assure that the library and the executing code MC_RUN connect efficiently with the World Wide Web application MC_Web as part of the Los Alamos Neutron Instrument Simulation Package (NISP). Since one of the most important features of MCLIB is its open structure and capability to incorporate any possible neutron transport or scattering algorithm, this document describes the current procedure that would be used by an outside user to add a feature to MCLIB. Details of the calling sequence of the core subroutine OPERATE are discussed, and questions of style are considered and additional guidelines given. Suggestions for standardization are solicited, as well as code for new algorithms.

1. Introduction

Monte Carlo is a method to integrate over a large number of variables. Random numbers are used to select a value for each variable, and the integrand is evaluated. The process is repeated a large number of times and the resulting values are averaged. For a neutron transport problem, we first select a neutron from the source distribution, and project it through the instrument using either deterministic or probabilistic algorithms to describe its interaction whenever it hits something. If it hits a detector, we tally it in a histogram representing where and when it was detected. This is intended to simulate the process of running an actual experiment (but it is *much* slower). Monte Carlo is a useful supplement to analytical treatment of an instrument, in particular to check and demonstrate "non-intuitive" focusing arrangements, but should never be used as a substitute for thinking. (We are grateful to Jack Carpenter for reminding us of this limitation of Monte Carlo.)

Keywords: MCLIB, Monte Carlo, simulation, instruments

The approach generally used in the MCLIB library routines is to treat the optical properties of neutron transport rather than microscopic nuclear interactions (although microscopic processes may be included in specific algorithms). The philosophy and structure of MCLIB (and of the executing program MC_RUN) were presented at ICANS-XIII [1], and that report, augmented by the proceedings of the 1996 Berkeley Workshop [2], has become the reference document for MCLIB. The current version may be accessed by anonymous ftp from

ftp://azoth.lansce.lanl.gov/pub/mclib/document

in three formats. Note that this document is updated frequently as features are added to the library. The source codes are also available through this ftp site.

The most exciting new features to report are the establishment of a Web application and standardization of the code under the name Neutron Instrument Simulation Package (NISP). That work is reported elsewhere in these proceedings [3], but all users and prospective users of NISP are urged to visit the web site at

http://bayberry.lanl.gov/lansce/Welcome.html

2. New Library Features and MC_RUN Updates

There are two new options for neutron sources. You may specify a file of individual neutron histories generated as a monitor output from a previous execution of MC_RUN, or you may specify a square (uniform) distribution of velocities.

A new region type is type 14, "toroidal mirror." Since the geometry of surfaces and regions is limited to quadratic, a torus can not be defined as a simple surface. This type illustrates how *any* form of geometry may be implemented *within* a region, which is itself bounded by quadratic surfaces.

The definitions of most sample types have been modified to allow the final directions of the neutrons to be limited in solid angle, for the purpose of variance reduction when detectors don't cover 4π. In particular, the isotropic scattering type 32 may have its solid angle defined by bands of direction cosines with respect to each axis. An auxiliary routine dOMEGA is provided to compute the resulting solid angle (if any!) for normalization. Sample types that are not isotropic (30, 34, and 36) can not be randomized in polar angle, but may have the azimuthal angle biased to illuminate specific detector geometries. Note that whenever solid-angle limits are used, multiple scattering is turned off. Another change in type 32 is that you may now specify a spectrum of δ-function energy changes instead of a single energy.

The encoding of two-dimensional detectors (type 43) now includes cylindrical and spherical coordinates as subtypes, as well as rectilinear and plane-polar. This makes the use of detectors with curved surfaces easier.

Many additions have been made to program MC_RUN to assist in debugging and to study "unexpected" events. A monitor (.MON) file may be written to record the passage of neutrons across a given surface (and this file may be used subsequently as a source file, see above). It is now possible to flag a surface preceding the monitor surface for correlation; then any neutron that crosses the correlation surface and subsequently reaches the monitor surface will be recorded in a .COR file. Another option is that the .COR file may record where the neutron was

immediately before reaching the surface being monitored. These files are direct-access binary, so special programs (such as SUPER_KNOW) must be adapted to extract the relevant information.

In response to a request at a recent workshop [4], a backtracking feature has been implemented. First, a neutron may be flagged as "bad" by some procedure during its transport, for example by coming through a chopper opening out of phase. If such a "bad" neutron reaches either the sample or any detector, then the complete history of surface and region crossings of that neutron will be written to a .BAD file in an ASCII format allowing subsequent study.

More variance-reduction features have been incorporated, also in response to the workshop [4]. In addition to the solid-angle biasing described above, up to 32 levels of splitting (or secondary neutron production) may be applied to a neutron. This includes multiple use of histories reaching the sample, or doubling when crossing flagged surfaces. User routines have the capability to split or create neutrons, as will be discussed below.

3. Program MC_RUN

A flow chart of the execution program MC_RUN is given in Fig. 1. There are three nested loops: the outermost loop is over the number of source neutrons (with a branch for stored and repeated neutrons); the next is transport between regions, determination of the subsequent region, and the interaction at the surface; and the innermost loop is what happens within any given region (subroutine OPERATE). All geometric relations between regions are handled in this main code, and all physics and algorithms of elements within regions are accessed through OPERATE, either

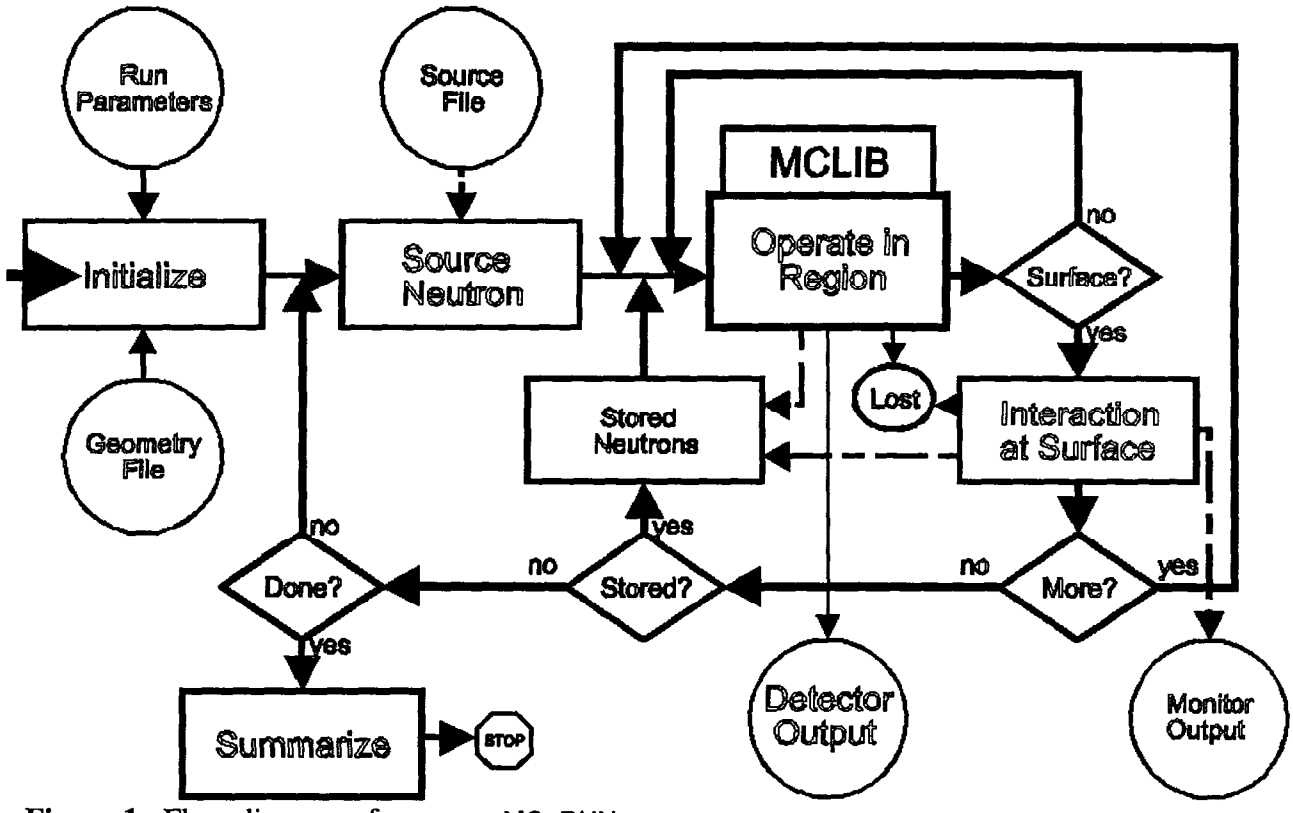

Figure 1. Flow diagram of program MC_RUN

as in-line code or as external procedures. The operation loop continues until the neutron reaches an exit surface of the region (or vanishes). Possible outputs of the operation include storing a created neutron for subsequent tracking, detection of the neutron, or absorption. It is within this loop that the MCLIB library is applied, and it is here that new features and modules may be inserted. This report is intended to assist in the incorporation of such new features.

4. Structures used in MCLIB / MC_RUN

The source codes for the Monte Carlo Library MCLIB and MC_RUN are written in a subset of ANSI-standard Fortran 90 (F90). To improve portability, only F90 features that are also included as "VAX extensions" to F77 are used (F77/VAX). Declaration statements use the F77 format, which is allowed in F90. Two exceptions to the F90 standard are that the character "." is used as the structure delimiter instead of "%", and that structures are defined and declared in STRUCTURE and RECORD statements instead of TYPE; these are again to improve portability to F77/VAX compilers.. F90 is a structured language, but not object oriented. Some concepts of object-oriented programming are used, but the emphasis is placed on execution speed.

The coordinate system used in MCLIB assumes that the beam axis is generally in the z-direction, the x-direction is to the particle's right as it travels in the z-direction, and the y-direction is always vertically upward (because gravity always acts in the negative y-direction). This left-handed coordinate system was chosen to match the X-Y coordinates of a plane detector in small-angle geometry. A module may rotate coordinates in the X-Z plane, but changing the direction

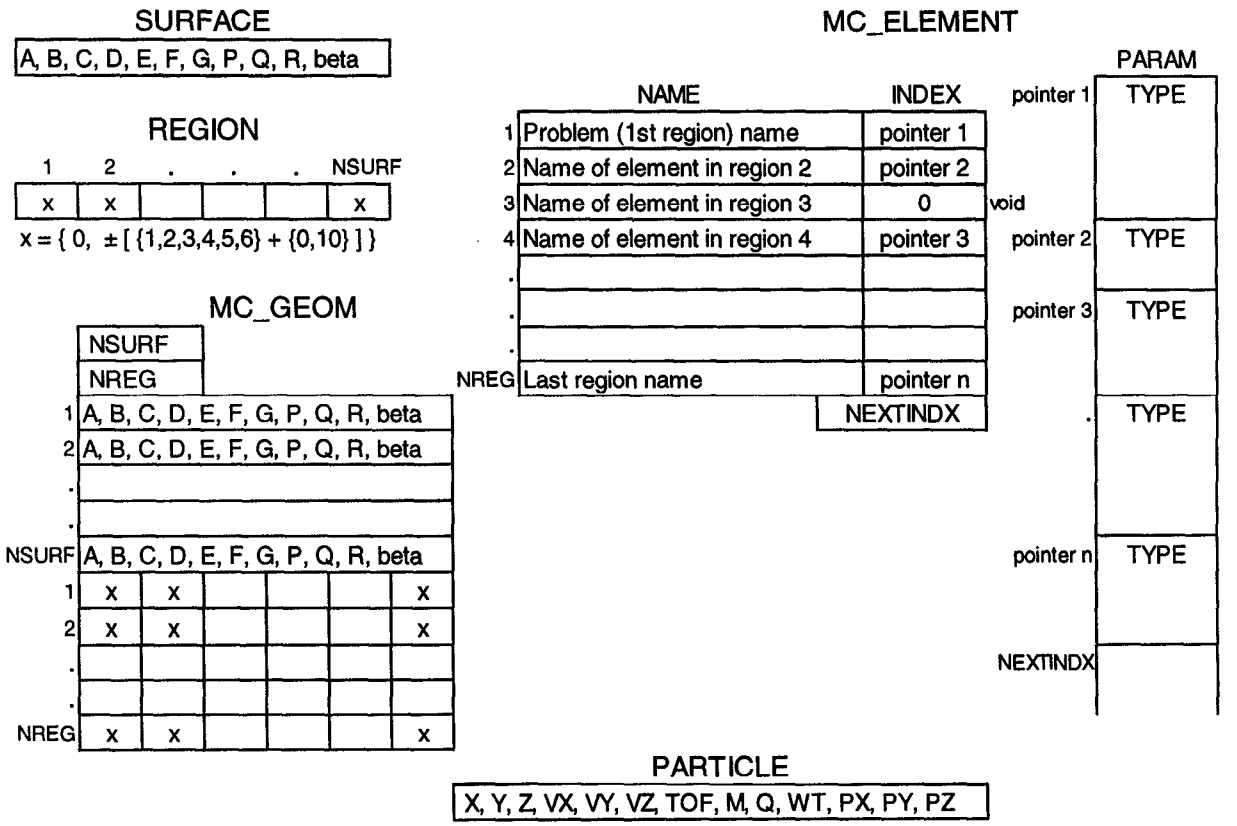

Figure 2. Structures used in MCLIB and MC_RUN

of gravity is strongly discouraged. The fundamental structures are PARTICLE, SURFACE, REGION, and MC_ELEMENT. These are illustrated in Fig. 2, and defined in an include file, MC_GEOM.INC, which is available with the source code at

ftp://azoth.lansce.lanl.gov/pub/mclib/fortran

All elements in PARTICLE and SURFACE structures are REAL*4. Compared to REAL*8, there is a significant reduction of the length of history files containing large numbers of particles, but there are some limitations in defining surfaces, which will be described below.

The elements of a PARTICLE structure are

(X, Y, Z) = position of the particle (m)
(VX, VY, VZ) = particle velocity (m/μs)
TOF = particle time-of-flight (μs)
M = atomic mass number, *e.g.*, 1 for a neutron or 0 for a photon
Q = atomic number of particle, *e.g.*, 0 for a neutron or +1 for a proton
WT = statistical weight of particle. A single tracked history may represent more or less than one neutron. This allows source weighting and the tracking of low-probability events for variance reduction.
(PX, PY, PZ) = average polarization vector of particle beam. The magnitude of the vector represents the degree of polarization.

A module may use and change any of the components of a particle sent to it. For instance, motion is accomplished by updating X, Y, Z, and TOF, or partial absorption by decreasing WT. A particle may be split by making a copy and apportioning the original WT between the two instances.

The elements of a SURFACE structure are the ten coefficients of a general quadratic surface and a parameter describing the surface roughness. The equation of a surface is

$$Ax^2 + Bx + Cy^2 + Dy + Ez^2 + Fz + G + Pxy + Qzy + Rzx = 0 \qquad (1)$$

The use of single-precision real numbers limits the definitions of surfaces with quadratic terms if they are far from the origin. This is not a severe problem if the origin of the coordinate system is placed at the sample location, since the quadratic surfaces used tend to be centered near the sample. The surface roughness parameter BETA is presently defined only for values between 0 and +1 as a long-range waviness. It is the maximum slope error in a cosine distribution, with *rms* ≈ 0.58 BETA for small values of BETA. We are exploring representation of surface irregularities on shorter length scales, which could be represented by using the sign of BETA as a flag, or by using values > 1. A module may use any predefined surfaces, and may create a surface for its internal use, but must not modify any existing surfaces. Any surface more complex than quadratic (*e.g.*, a toroid) can only be defined within a module.

A REGION structure is a vector IGEOM of signed integers (INTEGER*2) which give the relationship of the region to every surface. If IGEOM(JSURF) = 0, the region is not bounded by surface JSURF. If not zero, then the sign of the integer defines which side of the surface is "inside" the region by the sign of eq. (1) when evaluated at a point (x, y, z). That is, in order for a particle to be inside the region, the left side of eq. (1) must have the same sign as IGEOM for the surface. (If eq. (1) evaluates to 0, then 1st and 2nd derivatives will also be considered.) The numeric value of IGEOM is also significant:

1: ordinary surface described by roughness BETA, with possibility of refraction or critical

reflection depending on wavelength and relative index of refraction

2.: totally reflecting surface (from inside the region)

4: totally absorbing surface when hit from inside the region; *i.e.*, no exit

5: special conditions apply before exiting region; for instance, a coordinate transform may be required. Then treat as type 1. An example of this usage is given below.

6: treat as type 1, but split the particle into 2 equal instances after crossing

The value of IGEOM is increased by 10 if the surface is that of a region "embedded" within the region being defined (embedded and reentrant regions are discussed in the MCLIB Document). Modules have access to the region definitions, but should not have to deal with anything but the sign of IGEOM. No changes of region definitions are allowed in modules.

The structure MC_ELEMENT is most relevant when writing a module, because this is the structure used to define the contents of all regions. There is a one-to-one correspondence of elements and regions. An element has a 40-character NAME and an integer pointer INDEX into an array of REAL*4 parameters, PARAM. (A special case is void regions, which have INDEX = 0.) The number in the PARAM block at the location INDEX is the ELMNT_TYPE of the region; the integer part of the value is the type and fractions may be used for subtypes. Any number of parameters may be defined; for each defined type nn, there is also a parameter NUMBER_nn that is one more than the number of parameters defined. The author of a module must obtain or assign a type number (70 through 79 are available for *ad hoc* or development use), and must define variables with global names. (Global names are necessary so that the creator of the geometry file will have exactly the same definitions as the executing program!) Descriptive names are preferred, and a prefix may be used to identify the relevant module; documentation of the meanings of the variables is essential. The names are defined in PARAMETER statements as integer offsets in the PARAM block, counting from 0 at the location referenced by INDEX. The complete list of defined types is given in the include file MC_ELMNT.INC, found at the ftp site. A module may modify an entry in the PARAM block for its own later use, but *not* as a mechanism for returning a value to MC_RUN. Use of static local variables (SAVE statements) is preferred over modifying PARAM entries. The NAME variable may be used to pass a file name to the module. The communication between MC_RUN and the modules is described in the next section. Note that there are rough "classes" of modules, organized by decades of the type number. In particular, samples are in the range 30–39 and detectors in the range 40–49. Such classes share common variable names.

5. Subroutine OPERATE

This subroutine contains the "methods" for every type of element, either in-line or as external subroutine calls. The task of an author of a new module is to incorporate the new type into OPERATE by including a new case. For development, dummy routines for cases 70–79 are pre-linked, so that a user may substitute his own routine for the dummy one without having to modify OPERATE. For examples and to show the case structure, excerpts from the subroutine are listed in Appendix A. All of the routines in the MCLIB library as listed and described in the MCLIB document are available to use in support of new modules. The calling sequence of OPERATE contains arguments to support a variety of classes of elements; for instance, detectors need to return encoding information. The arguments in the calling sequence are listed in order in Table 1. Any of these may also be used as arguments to additional lower-level subroutines. All arguments are passed by reference, so the programmer has the responsibility to use care in changing values.

Table 1. Arguments in Calling Sequence of Subroutine OPERATE

Input	Name	Type	Description	Output
Neutron	PART	Structure	A PARTICLE structure. Position, velocity, statistical weight, & polarization may change depending on the case.	Updated Neutron
Escape Distance	EXDIST	REAL*4	Recompute if velocity changes. Will be 0 if the neutron is moved all the way to an exit surface.	New Escape Distance
Parameters of Region	PARAMS	REAL*4 (0:*)	From the PARAM array. First value is type number of the region, used in CASE structure.	
MC_GEOM Structure	GEOM	Structure	All surface and region geometry definitions.	
Region Number	IREG	INT*4	Modified if region contains subregions, *e.g.*, chopper or Soller-slit package.	Subregion Number
Entrance Surface	JSURF	INT*4	Surface neutron is on initially, or 0 if not on a surface.	
Exit Surface	KSURF	INT*4	Surface toward which the velocity points; updated if direction changes; set to negative if reflection occurs.	New Exit Surface
Name of Region	NAME	CHAR*40	The region name may be an external file name, *e.g.*, an S(α,β) file for inelastic scattering.	
Transmission Flag	TRANSMIT	LOGICAL	Used by sample class to know when to split neutron into scattered and non-scattered histories.	
	FLAG	LOGICAL	Set to .FALSE. if something is peculiar, *e.g.*, neutron in wrong chopper frame.	Bad-Neutron Flag
	PART_2	Structure	Typically used to save the "scattered" neutron independently from "transmitted."	Second Neutron
	DET_WT	REAL*4	Non-zero output identifies region as detector. Includes detector efficiency.	Detected Statistical Weight
	IX	INT*4	First coordinate of position-sensitive detector.	Detector Cell Number, IX
	IY	INT*4	Second coordinate (if 2D detector).	Detector Cell Number, IY
Random-Number Seed	ISEED	INT*4	A single pseudo-random sequence is used throughout the package.	Random-Number Seed

The first argument, PART, is the neutron being tracked. The set of values from the PARAM block that is specific to this element is passed as a vector PARAMS. Values may thus be referenced in the form PARAMS(ParameterName), *e.g.*,

```
NSIG = PARAMS(NSIGMA0) + PARAMS(NSIGMAV)/SQRT(PART.VX**2+PART.VY**2+PART.VZ**2)
```

(this and other examples will be found in context in Appendix A). Note that PARAMS is declared in OPERATE to begin with index 0 (the type number), and this is a useful convention to use in lower-level subroutines as well so that the indexing of locations in the block needs no offset.

When OPERATE is called, MC_RUN has already determined the distance EXDIST to escape from the region along the initial trajectory (including gravity), and this is passed as the second

argument (useful for determining attenuation). If the velocity vector (magnitude or direction) changes within the module, EXDIST must be recomputed by a call to DTOEX. This allows OPERATE to loop internally (instead of returning to MC_RUN at each step) for effects such as multiple scattering, but it means that the entire geometry structure (GEOM) and the region and surface numbers (IREG, JSURF, and KSURF) must also be passed as arguments. For an example of multiple scattering, see type 36 in Appendix A. By default, OPERATE will move the particle by the distance EXDIST and will therefore return with EXDIST = 0; to avoid this for a particular case, end the case with a RETURN statement instead of falling through. IREG and KSURF are also output parameters in special cases. The final example in this section shows the use of subregions, and type 13 (Appendix A) shows negative KSURF used as a flag for reflection. The development cases 70–74 have calling sequences without detector binning or instrument geometry, and cases 75–79 have the complete calling sequence of OPERATE.

The calling sequence includes two logical variables for state information. MC_RUN expects to track neutrons transmitted through a sample as well as scattered neutrons. The flag TRANSMIT is initially .TRUE. and is used to identify the first encounter of a neutron with the sample so that the code will generate a second neutron (PART_2). That neutron will be scattered on a subsequent entry when TRANSMIT = .FALSE. (see type 36 in Appendix A). Any call to OPERATE is allowed to generate a new PART_2. The output variable FLAG is a debugging aide. When a module recognizes that something about the neutron deserves attention, such as passing through a chopper in the wrong frame (see type 20 in Appendix A), FLAG is set to .FALSE.. If such a "bad" neutron subsequently hits a sample or a detector, then MC_RUN will tally it and (if requested) will also write its full history to a file. This feature may also be used in *ad hoc* versions of OPERATE to address rare occurrences.

All histogramming functions are performed by MC_RUN, but detector encoding is a function of a detector-class element. The statistical weight detected and one or two cell numbers are returned through the calling sequence respectively as DET_WT, IX, and IY. (The example in Appendix C, type 43, also shows how all of the arithmetic of a module may be moved to a separate procedure, in this case a call to DET_2D.) Time-of-flight slices, which may be non-linear and different for each detector, are computed in MC_RUN because they involve storing arrays of slice boundaries and (since dynamic memory allocation is not available in F77/VAX) all variable-length memory arrays are reserved for management in the main program. The final argument is ISEED, the seed of the random-number generator, which is propagated through all levels of subroutines in MCLIB so that a single pseudo-random number sequence is used.

An example of a complex region (containing subregions, and also using a coordinate transformation) is a multi-slit collimator, type 11 in Appendix A. This type encompasses Sollers, tapered Sollers, and benders. Instead of defining every surface and region explicitly, the package is defined by its external geometry and by the slit spacing, taper (if any), and bend angle (for a bender consisting of cylindrical slits). The code for type 11 uses the slit spacing to determine which slit number the neutron is entering, and then translates and rotates the coordinate system so that the neutron appears to be in the one slit which is defined. A logical variable identifying that the neutron is in this type of region and the parameters used for the coordinate transformation are stored in static local variables so that they will be available to undo the transform when the neutron exits the region. There may be as many as five subregions to describe one opening: the lumen, the coating materials on each side, and half of the substrate on

each side. (Alternatively if the center of the array is a blade instead of an opening, the subregions could be the substrate, the two coatings, and half of the lumen on each side of the substrate.) The code tests the five regions immediately following the type 11 region in the geometry definitions, sets IREG to the proper value, and executes a RETURN without moving the particle. The entrance and exit surfaces and also the outer longitudinal surfaces of the slit descriptive regions must be tagged in the REGION structure with a value of 5 so that MC_RUN will call EXIT_REG (a second entry point in OPERATE) when a neutron crosses. This entry point is shown at the end of Appendix C, including the code relating to type 11. There are two situations. If the particle is still within the exterior geometry of the slit package (*i.e.*, still within the bounds of the type 11 region), then it must be crossing from one slit to another within the package. The slit number is changed and the coordinates transformed again so that it is once more entering the defined opening. If on the other hand the neutron is leaving the whole package, then the coordinate system is reconverted to match the outside world.
This example also shows how a permanent change of the coordinate system may occur in an element. If the type 11 region was a bender, then the z-axis is rotated. Another example of rotation in the X-Z plane is a crystal monochromator (see type 13 in Appendix A).

6. Style and Guidelines

Appendix A may also be used as a style guide for writing code for inclusion in MCLIB. By nature, questions of "style" are arbitrary, but some level of consistency will assist later generations in maintaining the code package. Upper case is preferred except for variable names that make more sense with mixed or lower case characters. Continuation lines should be written to be compatible with either fixed or floating format Fortran, by placing "&" in column 73 of the line to be continued and another "&" in column 6 of the continuation line. Always use IMPLICIT NONE before the INCLUDE statements or before any declaration statements, and declare every variable. DO and IF blocks should be indented by 3 characters. Numbered statements should be avoided. Lots of comments throughout, and enough spaces in code lines to see where the operators are. Make me jealous by writing code that is easier to read and to understand than mine (as well as being correct, of course).

All lines between the comment lines "C++" and "C—" will be included in the subroutine abstracts in Appendix B of the MCLIB Document. The "required" information includes a brief description of the purpose and methods used in the procedure, the original author and date and any references to publications or other sources of algorithms, the update history, any INCLUDE statements, descriptions and Fortran declarations of all variables in the calling sequence, and a list of any external routines called and declarations for those which are functions. For more examples, see the MCLIB Document. The complete source codes are also available (in 3 file formats) at the anonymous ftp site, *ftp://azoth.lansce.lanl.gov/pub/mclib/fortran.*

Physical and mathematical constants should be defined and placed in PARAMETER statements at the beginning of the procedure, or better yet given global names and included in the file CONSTANT.INC so that everyone will be using the same values. The present quantities in CONSTANT.INC are

GOVER2	= half the acceleration of gravity	=	4.858×10^{-12} m/μs^2
HOVERM	= Plank's constant/neutron mass	=	0.0039560339 m-Å/μs
HSQOV2M	= Plank's constant squared over 2 Mneutron	=	0.0818145347 eV-Å^2

PRECES_N	= neutron magnetic-moment precession rate factor	
	= $2\pi\mu_n M_n/h^2$	= 23160.451 radian/T/m/Å
ROOTM_2	= square root of half the neutron mass	= 72.298 $eV^{0.5}$-μs/m
TWOPI	= 2π	= 6.283185307

The units used throughout MCLIB are distance in m, time in μs, wavelength in Å, and energy in eV.

The examples in the previous section of this report describe operations within a region. It may be desired also to consider refraction and reflection when crossing the surface of a region, and this requires a method to determine the index of refraction (or equivalently the scattering-length density). Presently only two element types (amorphous materials and supermirrors) provide this information, through the complex function GET_RHO. Addition of other types to this function should be straightforward.

7. Conclusion

This report has described the current status of the structure of the MCLIB library, and the procedure by which new features can be added to the low-level code. We propose this as a standard. However, many features are arbitrary accidents of history, and we are requesting input from users to determine what *should* be changed and what *must* be changed before the standard is established. Although we have tried to describe the functionality requirements that led to the present status, it must be expected that improvements will be made both from the point of view of computer science and also run-time efficiency (which we take to be the ultimate test). Please address any questions and suggestions to PASeeger@aol.com.

Acknowledgments

This effort has been supported historically by the Los Alamos National Laboratory and the Manuel Lujan Jr. Neutron Scattering Center, a national user facility funded by the United States Department of Energy, Office of Basic Energy Sciences-Materials Science, under contract number W-7405-ENG-36 with the University of California.

References

[1] P. A. Seeger, "The MCLIB Library: Monte Carlo simulation of neutron scattering instruments," 13th Meeting of the International Collaboration on Advanced Neutron Sources, October 11–14, 1995, Paul Scherrer Institut, Villigen, Switzerland, PSI Proceedings 95-02, pp. 194–212.

[2] P. A. Seeger, "The MCLIB Library: new features," Workshop on Methods of Neutron Scattering Instrument Design," Lawrence Berkeley Laboratory, Berkeley, CA, Sept 23–27, 1996.

[3] T. G. Thelliez, L. L. Daemen, P. A. Seeger, and R. P. Hjelm, Jr., "MC-Web: A WWW-based application for NISP," 14th Meeting of the International Collaboration on Advanced Neutron Sources, June 14–19, 1998, Starved Rock Lodge, IL (these proceedings).

[4] R. K. Crawford, "Report on the Workshop on Monte Carlo Simulation of Neutron Scattering Instruments," Nov. 13–14, 1997, Argonne National Laboratory.

Appendix A. Excerpts from Subroutine OPERATE

```
C++
C********** O P E R A T E **********
C********** E X I T _ R E G **********
C
      SUBROUTINE OPERATE(PART, EXDIST, PARAMS, GEOM, IREG, JSURF, KSURF,&
     &                   NAME, TRANSMIT, FLAG, PART_2, DET_WT,          &
     &                   IX, IY, ISEED)
C
C               EXIT_REG(PART, GEOM, IREG, JSURF)
C
C  Routines to operate on a particle within (or exiting from) a region
C    containing material, collimation elements, time-dependent devices,
C    samples, or detectors.  Included region types/actions are
C         1    amorphous unpolarized material: move to exit w/reduced weight
C         2    aluminum: move to exit w/reduced weight
C         5    beryllium: move to exit w/reduced weight
C         6    single-crystal filter: move to exit w/reduced weight
C        11    multi-slit collimator, vertical: selects subregion w/o moving
C        12    multi-slit collimator, horizontal: selects subregion w/o moving
C        13    crystal monochromator: reflect from surface or move to exit, and
C              rotate coordinates
C        14    toroidal mirror: region divided into subregions by toroidal
C              mirror: move to exit w/reduced weight
C        20    blade or disk chopper: move to exit OR select subregion
C        22    gravity focuser: selects subregion without moving
C        23    removable beamstop: set weight=0 if not transmission mode
C        30    fixed-Q or hard-sphere scatterer: transmitted or scattered
C              particle moves to exit with modified weight
C        32    isotropic scatterer with spectrum of energy changes: transmitted
C              or scattered particle is moved to exit with modified weight
C        34    inelastic scatter using S(alpha,beta) from MCNP: transmitted or
C              scattered particle is moved to exit with modified weight
C        35    reflectometry, multilayer: reflect from surface w/o moving,
C              same weight if transmitted or reduced weight if scattered
C        36    general powder scatterer: transmitted or scattered particle moves
C              to exit with modified weight
C        40    single detector: determine detection probability
C        41    linear detector, vertical: determine y-bin and probability
C        42    linear detector, transverse: determine x-bin and probability
C        43    2-dimensional detector: determine x- and y-bins and probability
C        44    linear detector, longitudinal: determine x-bin and probability
C        90    source size and phase space: weight=0 if outside surface
C
C  P. A. Seeger, April 20, 1994
C    . . . (modification history)
C    13 May 1998: subtype 32.2, 30.1, 30.2, 34.1, 36.1 [LLD,PAS]
C
C  Definitions of STRUCTUREs:
      IMPLICIT NONE
      INCLUDE 'mc_geom.inc'
      INCLUDE 'mc_elmnt.inc'
      INCLUDE 'constant.inc'
C
C  Variables in calling sequence:
C    PART = record containing description of particle (input/output)
C    EXDIST = distance to exit surface particle is aimed at (m) (input/output)
```

```
C     PARAMS = array with description of what is in this region (input)
C     GEOM = structure with all surface and region definitions (input)
C     IREG = region number of device, or subregion within device (input/output)
C     JSURF = surface number, if particle is initially on surface (input)
C     KSURF = surface number that particle is pointed toward (input/output)
C     NAME = name of region, used as file name for type 34 (input)
C     TRANSMIT = flag to compute transmission of sample types 30-39 (input)
C     FLAG = flag set to .FALSE. if (e.g.) chopper in wrong frame (output)
C     PART_2 = description of particle created by operation (output)
C     DET_WT = statistical weight of detected particle (output)
C     IX, IY = position bin numbers of detected particle (output)
C     ISEED = random-number generator seed (input/output)
      RECORD    /PARTICLE/ PART, PART_2
      RECORD    /MC_GEOM/  GEOM
      REAL*4    PARAMS(0:*), EXDIST, DET_WT
      INTEGER   IREG, JSURF, KSURF, IX, IY, ISEED
      CHARACTER NAME*40
      LOGICAL   TRANSMIT, FLAG
C
C  Externals:
C    ANGLI     ANGTORUS  ATTEN_Al  ATTEN_Be  ATTEN_X   DET_2D    DISTORUS
C    DTOEX     ELSCAT    GET_RHO   GRAV_FOC  KERNEL    LMONOCRM  LORRAND
C    LREFLCT   MOVEX     NEXTRG    ORRAND    PLEXP     PLNORM    PLQSPHR
C    POWDER    RAN       REFLAYER  RFLN      SNELL     TESTIN    XCHOPPER
      REAL*4    ATTEN_Al, ATTEN_Be, ATTEN_X, DET_2D, DISTORUS, GRAV_FOC,&
     &          PLEXP, PLNORM, PLQSPHR, RAN, REFLAYER, XCHOPPER
      COMPLEX*8 GET_RHO
      INTEGER   NEXTRG
      LOGICAL   LMONOCRM, LORRAND, LREFLCT, TESTIN
C--
C  Local variables:
      REAL*4  NSIG, LAMBDA, XC, YGF, X, Y, D, TRANPROB, ATTEN,          &
     &        ALPHA, SLIT, DELTA, TAPER, ZENTER, SIN_PHI, COS_PHI,      &
     &        COS_TH, SIN_TH, V, AP, BP, CP, REFPROB, F, Q, KZ, EN0,    &
     &        EN1, EN2, TWOSINTH, SIGSCAT, SIGABS, S_MULT, RATIO
      COMPLEX CXRATIO
      INTEGER I, ITYPE, NREG, SUBTYPE, JINDX, JTYPE, KINDX, KTYPE
      LOGICAL LOPENING, LGF, LMULTX, LMULTY, LOPEN, DONE, VERTICAL,     &
     &        INSIDE
C
      SAVE    LGF, LMULTX, LMULTY, YGF, SLIT, DELTA, TAPER,             &
     &        ZENTER, SIN_PHI, COS_PHI, NREG, ATTEN, ALPHA, LAMBDA, Q,  &
     &        EN0, TWOSINTH, TRANPROB
      DATA    LGF,    LMULTX, LMULTY, YGF,SLIT/                         &
     &        .FALSE.,.FALSE.,.FALSE.,0., 0.  /
C
      PART_2.WT = 0.
      DET_WT = 0.
      FLAG = .TRUE.
C
      ITYPE = PARAMS(0)
      IF (ITYPE .EQ. 0) THEN
C        Material is total absorber
         EXDIST = 0.
         PART.WT = 0.
C
      ELSE IF (ITYPE .EQ. 1) THEN
C        Material is amorphous unpolarized
         NSIG = PARAMS(NSIGMA0) + PARAMS(NSIGMAV)/SQRT(PART.VX**2 +     &
```

```
     &                                          PART.VY**2 + PART.VZ**2)
         IF (NSIG .GT. 0.) THEN
            IF (NSIG*EXDIST .LT. 12.) THEN
               PART.WT = PART.WT * EXP(-EXDIST*NSIG)
            ELSE
               PART.WT = 0.
            END IF
         END IF
C
       ELSE IF (ITYPE .EQ. 11) THEN
C        Region contains a multi-slit collimator with vertical blades
         LMULTX = .TRUE.
C        Save parameters to use when exiting region
         NREG    = IREG
         DELTA   = PARAMS(C_DELTA)
         TAPER   = PARAMS(C_TAPER)
         ZENTER  = PARAMS(C_ZENTER)
         SIN_PHI = PARAMS(B_SIN_PHI)
         COS_PHI = PARAMS(B_COS_PHI)
C        Translate particle into central slit
         IF (TAPER.EQ.0. .OR. PART.Z.EQ.ZENTER) THEN
            SLIT = ANINT(PART.X / DELTA)
            PART.X = PART.X - SLIT*DELTA
         ELSE
C           Need to account for taper when determining slit number
            SLIT = ANINT(PART.X / (DELTA-TAPER*(PART.Z-ZENTER)))
            PART.X = PART.X-SLIT*(DELTA-TAPER*(PART.Z-ZENTER))
         END IF
         IF (TAPER .NE. 0.) THEN
C           Tapered slits, also need to rotate particle velocity vector
            COS_TH = 1.-0.5*(SLIT*TAPER)**2
            SIN_TH = SLIT*TAPER*COS_TH
            V = PART.VX
            PART.VX =  COS_TH*V + SIN_TH*PART.VZ
            PART.VZ = -SIN_TH*V + COS_TH*PART.VZ
         END IF
         IF (SIN_PHI .NE. 0.) THEN
C           Element is a bender; rotate velocity vector half the angle
            V = PART.VX
            IF (PART.VZ .GT. 0.) THEN
C              Entering in proper direction, rotate CCW (if phi > 0.)
               PART.VX =  COS_PHI*V - SIN_PHI*PART.VZ
               PART.VZ =  SIN_PHI*V + COS_PHI*PART.VZ
            ELSE
C              Entering backwards, shift CW instead of CCW
               PART.VX =  COS_PHI*V + SIN_PHI*PART.VZ
               PART.VZ = -SIN_PHI*V + COS_PHI*PART.VZ
            END IF
         END IF
C        May be as many a five sub-regions; find which one
         KSURF = JSURF
         DO I=1,5
            IF (TESTIN(PART, GEOM, IREG+I, JSURF)) THEN
               IREG = IREG+I
               RETURN
            END IF
         END DO
         IREG = 0
         RETURN
```

```
C
      ELSE IF (ITYPE .EQ. 13) THEN
C        Region is a crystal monochromator with mosaic spread
         IF (LMONOCRM(PART, PARAMS, GEOM.SURFACE(JSURF),                &
     &                COS_TH, AP, BP, CP, ISEED)) THEN
            CALL RFLN(PART, COS_TH, AP, BP, CP)
            EXDIST = 0.
C           Still on same surface, flag reflection with - sign
            KSURF = - JSURF
         ELSE
C           No reflection; move to exit surface before rotating coordinates
            CALL MOVEX(PART, EXDIST)
         END IF
         IF (PARAMS(M_SIN_2TH) .NE. 0.) THEN
C           Redefine instrument axis; rotate velocity vector
            X = PART.X
            V = PART.VX
            IF (PART.VZ .GT. 0.) THEN
C              Moving in proper direction, rotate CCW (if 2theta > 0.)
               PART.X =  PARAMS(M_COS_2TH)*X                            &
     &                  -PARAMS(M_SIN_2TH)*(PART.Z - PARAMS(M_Z0))
               PART.Z =  PARAMS(M_Z0) + PARAMS(M_SIN_2TH)*X             &
     &                  +PARAMS(M_COS_2TH)*(PART.Z - PARAMS(M_Z0))
               PART.VX = PARAMS(M_COS_2TH)*V-PARAMS(M_SIN_2TH)*PART.VZ
               PART.VZ = PARAMS(M_SIN_2TH)*V+PARAMS(M_COS_2TH)*PART.VZ
            ELSE
C              Moving backwards, shift CW instead of CCW
               PART.X =  PARAMS(M_COS_2TH)*X                            &
     &                  +PARAMS(M_SIN_2TH)*(PART.Z - PARAMS(M_Z0))
               PART.Z =  PARAMS(M_Z0) - PARAMS(M_SIN_2TH)*X             &
     &                  +PARAMS(M_COS_2TH)*(PART.Z - PARAMS(M_Z0))
               PART.VX = PARAMS(M_COS_2TH)*V+PARAMS(M_SIN_2TH)*PART.VZ
               PART.VZ = PARAMS(M_SIN_2TH)*V+PARAMS(M_COS_2TH)*PART.VZ
            END IF
         END IF
C
      ELSE IF (ITYPE .EQ. 20) THEN
C        Region contains a disk or blade chopper; find location of edge
         XC = XCHOPPER(PART.TOF,PARAMS(CHP_OPEN),PARAMS(CHP_CLOSE),     &
     &                 PARAMS(CHP_JITTER), PARAMS(CHP_VEL),             &
     &                 PARAMS(CHP_PERIOD), LOPENING, ISEED)
         SUBTYPE = NINT(10.*(PARAMS(0) - 20.0))
         IF (IAND(SUBTYPE,1) .EQ. 0) THEN
C           Chopper is moving horizontally, compare to PART.X
            X = PART.X
         ELSE
C           Chopper is moving vertically, compare to PART.Y
            X = PART.Y
         END IF
         LOPEN = ((X .LT. XC) .XOR. (PARAMS(CHP_VEL) .GT. 0.))          &
     &           .XOR. LOPENING
C        If counter-rotating chopper, also test absolute values
         IF (SUBTYPE.GE.2 .AND. ABS(X).GT.ABS(XC)) LOPEN = .FALSE.
         IF (LOPEN) THEN
            FLAG = NINT((PART.TOF-.5*(PARAMS(CHP_OPEN) +                &
     &              PARAMS(CHP_CLOSE))) / PARAMS(CHP_PERIOD)) .EQ. 0
         ELSE
C           Didn't make it through this chopper opening
            KSURF = JSURF
```

```
            IF (TESTIN(PART, GEOM, IREG+1, JSURF)) THEN
C              Next region is description of chopper blade material
               IREG = IREG+1
               RETURN
            ELSE
C              Chopper blade is opaque absorber
               EXDIST = 0.
               PART.WT = 0.
            END IF
         END IF
C
       ELSE IF (ITYPE .EQ. 36) THEN
C        Powder sample with Bragg scattering
         IF (TRANSMIT) THEN
C           Save incident particle and wavelength for scattering
            PART_2 = PART
            LAMBDA = HOVERM / SQRT(PART.VX**2 + PART.VY**2 + PART.VZ**2)
C           Transmitted particle has reduced weight, get attenuation terms
            CALL POWDER(PARAMS, LAMBDA, SIN_TH, SIGSCAT, SIGABS, ISEED)
            ALPHA = 100.*(SIGSCAT + SIGABS)
            TRANPROB = EXP(-ALPHA*EXDIST)
            PART.WT = PART.WT * TRANPROB
C           Scattered (or absorbed) particles have the rest of the weight
            PART_2.WT = PART_2.WT * (1. - TRANPROB) *                    &
     &                  SIGSCAT/(SIGSCAT+SIGABS)
         ELSE
C           Prepare for multiple scattering; where was the interaction?
            D = PLEXP(ALPHA, EXDIST, ISEED)
            CALL MOVEX(PART, D)
C           Choose angle from possible Bragg angles
            CALL POWDER(PARAMS, LAMBDA, SIN_TH, SIGSCAT, SIGABS, ISEED)
C           Find new velocity vector
            TWOSINTH = 2.*SIN_TH
            IF (IAND(SUBTYPE,1) .EQ. 0) THEN
C              Random azimuthal angle in 2 pi
               CALL ELSCAT(PART.VX, PART.VY, PART.VZ, TWOSINTH, ISEED)
C              Determine probability of scattering again
               D = PLEXP(ALPHA, 0., ISEED)
            ELSE
C              Limit the azimuthal angle
               CALL ELSCAT2(PART.VX, PART.VY, PART.VZ, TWOSINTH,         &
     &                      PARAMS(PHI_MIN),PARAMS(PHI_MAX),FLAG,ISEED)
               PART.WT = PART.WT * (PARAMS(PHI_MAX) - PARAMS(PHI_MIN))   &
     &                           /TWOPI
C              No multiple scattering in this case
               D = 0.
            END IF
C           Determine probability of scattering again before escaping region
            CALL DTOEX(PART, GEOM, IREG, 0, KSURF, EXDIST)
            DO WHILE (D.LT.EXDIST .AND. D.GT.0.)
               CALL MOVEX(PART, D)
C              Get another Bragg angle (same attenuation length)
               CALL POWDER(PARAMS, LAMBDA, SIN_TH, SIGSCAT, SIGABS,      &
     &                     ISEED)
               IF (RAN(ISEED) .LT. SIGABS/(SIGSCAT+SIGABS)) THEN
C                 Particle has been absorbed
                  PART.WT = 0.
                  D = 0.
               ELSE
```

```
                  TWOSINTH = 2.*SIN_TH
                  CALL ELSCAT(PART.VX, PART.VY, PART.VZ, TWOSINTH,ISEED)
                  CALL DTOEX(PART, GEOM, IREG, 0, KSURF, EXDIST)
                  D = PLEXP(ALPHA, 0., ISEED)
               END IF
            END DO
         END IF
C
      ELSE IF (ITYPE .EQ. 43) THEN
C        Detector, 2-dimensional position sensitive
         DET_WT = DET_2D(PART, PARAMS, IX, IY, ISEED)
C
      ELSE IF (ITYPE .EQ. 70) THEN
         CALL TYPE_70(PART, EXDIST, PARAMS, NAME, TRANSMIT, FLAG,       &
     &                PART_2, ISEED)
         RETURN
C
      ELSE IF (ITYPE .EQ. 75) THEN
         CALL TYPE_75(PART, EXDIST, PARAMS, GEOM, IREG, JSURF, KSURF,   &
     &              NAME, TRANSMIT, FLAG, PART_2, DET_WT, IX, IY, ISEED)
C
      END IF
C
C     Move the particle
      IF (EXDIST .GT. 0.) CALL MOVEX(PART, EXDIST)
C
      IF (DET_WT .NE. 0.) THEN
C        Particle was detected, so check detector efficiency
         IF (PARAMS(D_ALPHA_1A) .GT. 0.) THEN
            LAMBDA = HOVERM / SQRT(PART.VX**2 + PART.VY**2 + PART.VZ**2)
            DET_WT = DET_WT*(1. - EXP(-LAMBDA*PARAMS(D_ALPHA_1A)))
         END IF
         PART.WT = PART.WT - DET_WT
      END IF
C
      RETURN
C
C     Need special actions when leaving some regions
C
      ENTRY EXIT_REG(PART, GEOM, IREG, JSURF)
C
      IF (LMULTX) THEN
C        Crossing a surface in a multi-slit (vertical) device
         IF (NEXTRG(PART, GEOM, NREG, JSURF) .EQ. NREG) THEN
C           Still within device, shifting to neighboring slit
            IF (PART.X .GT. 0.) THEN
               D = +1.
            ELSE
               D = -1.
            END IF
            SLIT = SLIT + D
            PART.X = PART.X - D*DELTA
            IF (TAPER .NE. 0.) THEN
               PART.X = PART.X + D*TAPER*(PART.Z - ZENTER)
               COS_TH = 1. - 0.5*TAPER**2
               SIN_TH = D*TAPER*COS_TH
               V = PART.VX
               PART.VX =  COS_TH*V + SIN_TH*PART.VZ
               PART.VZ = -SIN_TH*V + COS_TH*PART.VZ
```

```
            END IF
          ELSE
C            Actually exiting from device, restore X, VX, and VZ
             PART.X = PART.X + SLIT*DELTA
             IF (TAPER .NE. 0.) THEN
C               Tapered slits, exit not at same offset as entrance
                PART.X = PART.X - SLIT*TAPER*(PART.Z - ZENTER)
C               Also need to rotate particle velocity vector
                COS_TH = 1.-0.5*(SLIT*TAPER)**2
                SIN_TH = SLIT*TAPER*COS_TH
                V = PART.VX
                PART.VX = COS_TH*V - SIN_TH*PART.VZ
                PART.VZ = SIN_TH*V + COS_TH*PART.VZ
             END IF
             IF (SIN_PHI .NE. 0.) THEN
C               Element is a bender; rotate by the second half of the angle
                V = PART.VX
                IF (PART.VZ .GT. 0.) THEN
C                  Exiting in proper direction, rotate CCW (if phi > 0.)
                   PART.VX =  COS_PHI*V - SIN_PHI*PART.VZ
                   PART.VZ =  SIN_PHI*V + COS_PHI*PART.VZ
                ELSE
C                  Exiting backwards, rotate CW instead of CCW
                   PART.VX =  COS_PHI*V + SIN_PHI*PART.VZ
                   PART.VZ = -SIN_PHI*V + COS_PHI*PART.VZ
                END IF
             END IF
             LMULTX = .FALSE.
          END IF
C
       END IF
       RETURN
       END
```