SLAC PUB-4791
November 1988
(N)

17. Geometry Methods and Packages*

Walter R. Nelson and Theodore M. Jenkins

Radiation Physics Group
Stanford Linear Accelerator Center
Stanford, California 94309, U.S.A.

17.1 MATHEMATICAL CONSIDERATIONS

The trajectory of a particle in a Monte Carlo calculation can be described by position and direction vectors

$$\vec{X} = x\hat{i} + y\hat{j} + z\hat{k} \tag{17.1}$$

and

$$\hat{U} = u\hat{i} + v\hat{j} + w\hat{k} , \tag{17.2}$$

respectively, where (x, y, z) are the coordinates of the particle at point $P(x, y, z)$ (*e.g.*, see figures below), and (u, v, w) are its direction cosines (the symbol $\hat{}$ denotes a *unit vector*). These quantities, together with such things as particle type, energy, weight, time, etc., define the *state function* of the particle.

In general, one is interested in determining the point of intersection $P'(x', y', z')$ of the vector $t\hat{U}$ with any given surface that constitutes the geometry for the problem at hand. To be more specific, the distance $t = |\overrightarrow{PP'}|$ is generally needed in order to compare with the actual transport distance that is about to be used in the simulation. In the following sections, we will develop the basic mathematical expressions for the intersection of the particle trajectories with plane and conic surfaces (*i.e.*, cylinders, cones, and spheres). We will show how they are used in the EGS4 Code System[1], which should be typical of the way it is done elsewhere. The remainder of this chapter will then be devoted to a general survey of some of the more prominent geometry packages currently being used in electron-photon Monte Carlo.

17.1.1 Intersection of a vector with a plane surface.

A plane surface can be described by the vector to a point P'' on the surface

$$\vec{C} = c_1\hat{i} + c_2\hat{j} + c_3\hat{k} \tag{17.3}$$

and a unit vector normal to the surface

$$\hat{N} = n_1\hat{i} + n_2\hat{j} + n_3\hat{k} , \tag{17.4}$$

* Work supported in part by the Department of Energy, contract DE-AC03-76SF00515.

Chapter 17 (Pages 385-405) in "Monte Carlo Transport of Electrons and Photons," T.M. Jenkins, W.R. Nelson, and A. Rindi, Editors (Plenum Publishing Co. 1988)

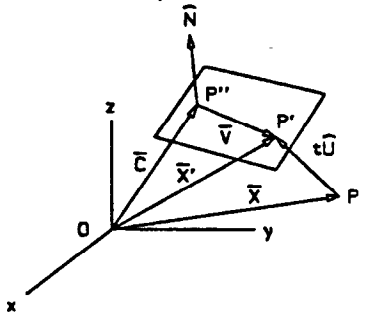

Figure 17.1. Vector diagram of particle intersecting a plane surface.

as shown in Fig. 17.1.

The condition for the intersection point P' to lie on the plane is obtained from the vector dot product

$$\vec{V} \cdot \hat{N} = 0 \,. \tag{17.5}$$

From the diagram we see that

$$\vec{V} = \vec{X}' - \vec{C} = \vec{X} + t\hat{U} - \vec{C} \,, \tag{17.6}$$

which leads to the solution

$$t = \frac{(\vec{C} - \vec{X}) \cdot \hat{N}}{\hat{U} \cdot \hat{N}} \,. \tag{17.7}$$

Equation 17.7 is indeterminant when $\hat{U} \cdot \hat{N} = 0$, corresponding to the physical situation in which the particle travels *parallel* to the plane. The particle travels *towards* the plane when $t > 0$ and *away* from it when $t < 0$. Expanded into its components, Eqn. 17.7 becomes

$$t = \frac{(c_1 - x)n_1 + (c_2 - y)n_2 + (c_3 - z)n_3}{un_1 + vn_2 + wn_3} \,. \tag{17.8}$$

17.1.2 The PLANE1 algorithm available in EGS4.

SUBROUTINE PLANE1 (see Fig. 17.2) of the EGS4 Code System provides an example of an algorithm for determining the intersection of a particle trajectory with a plane surface using the equations developed above. The components of $\vec{X}$ and $\hat{U}$ are available through COMMON/STACK/. NP is the *stack pointer—i.e.*, the particle currently being followed—of which there can be as many as 40 in the default version of EGS4. The vectors that define the plane, $\vec{C}$ and $\hat{N}$, are described by arrays PCOORD and PNORM, respectively, and are passed in COMMON/PLADTA/ (for a maximum of 100 planes in this example).

Except for parameter INPT, which allows for more efficient determination of $t > 0$ ("hit") versus $t < 0$ ("miss"), the algorithm in PLANE1 is based precisely on the equations developed in the previous section.

17.1.3 Intersection of a vector with a cylindrical surface.

The example we will use is a right circular cylinder of radius R whose axis of rotation lies along the z-axis as shown in Fig. 17.3 . The intersection point P' is located by

```
SUBROUTINE PLANE1(IPLN,INPT,IHIT,TPLN);
"-------------------------------------------------------------------------"
"  Input:  IPLN      Plane identification number                          "
"          INPT = 1  Surface normal points AWAY from current region       "
"               =-1  Surface normal points TOWARDS current region         "
"  Output: IHIT = 0  Particle travels AWAY from surface (a miss)          "
"               = 1  Particle travels TOWARDS surface (a hit)             "
"               = 2  Particle travels PARALLEL TO surface (a miss)        "
"          TPLN      Distance to surface (when IHIT=1)                    "
"-------------------------------------------------------------------------"
COMMON/STACK/E(40),X(40),Y(40),Z(40),U(40),V(40),W(40),DNEAR(40),
         WT(40),IQ(40),IR(40),NP;
DOUBLE PRECISION E;
COMMON/PLADTA/PCOORD(3,100),PNORM(3,100);

UDOTN=PNORM(1,IPLN)*U(NP) + PNORM(2,IPLN)*V(NP) + PNORM(3,IPLN)*W(NP);
UDOTNP=UDOTN*INPT;

IF (UDOTNP.EQ.0.0) [ IHIT=2; "Parallel to z-axis (indeterminant)" ]
ELSEIF (UDOTNP.LT.0.0) [ IHIT=0; "Traveling away from surface"]
ELSE [ "Traveling towards surface---determine distance (TPLN)"
  IHIT=1;
  TPLN=PNORM(1,IPLN)*(PCOORD(1,IPLN)-X(NP))
         + PNORM(2,IPLN)*(PCOORD(2,IPLN)-Y(NP))
            + PNORM(3,IPLN)*(PCOORD(3,IPLN)-Z(NP));
  TPLN=TPLN/UDOTN;
  ]

     RETURN;
     END;
```

Figure 17.2. Listing of SUBROUTINE PLANE1.

vector $\vec{\mathrm{X}}'$, whose $\hat{\mathrm{k}}$-component is defined by

$$\vec{\mathrm{V}} = (\vec{\mathrm{X}}' \cdot \hat{\mathrm{k}})\hat{\mathrm{k}} = (z + tw)\hat{\mathrm{k}} \,, \tag{17.9}$$

since

$$\vec{\mathrm{X}}' = \vec{\mathrm{X}} + t\hat{\mathrm{U}} = (x + tu)\hat{\mathrm{i}} + (y + tv)\hat{\mathrm{j}} + (z + tw)\hat{\mathrm{k}} \,. \tag{17.10}$$

The radial vector

$$\vec{\mathrm{R}} = \vec{\mathrm{X}}' - \vec{\mathrm{V}} = (x + tu)\hat{\mathrm{i}} + (y + tv)\hat{\mathrm{j}} \tag{17.11}$$

can then be squared (*i.e.*, $\vec{\mathrm{R}} \cdot \vec{\mathrm{R}}$) to give the quadratic equation

$$(u^2 + v^2)t^2 + 2(xu + yv)t + (x^2 + y^2 - R^2) = 0 \,. \tag{17.12}$$

The solution of this equation can be written

$$t = \frac{-\beta \pm \sqrt{\beta^2 - \alpha\gamma}}{\alpha} \,, \tag{17.13}$$

where

$$\begin{aligned} \alpha &= u^2 + v^2 \,, \\ \beta &= xu + yv \,, \\ \gamma &= x^2 + y^2 - R^2 \,. \end{aligned} \tag{17.14}$$

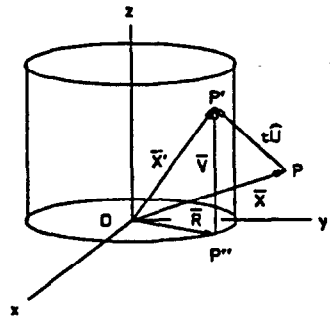

Figure 17.3. Vector diagram of particle intersecting a cylindrical surface.

A few special cases are worth discussing. First, since

$$u^2 + v^2 + w^2 = 1 \,, \tag{17.15}$$

$\alpha = 0$ implies that $w = \pm 1$, so that the particle travels parallel to the cylinder surface and does not intersect it (except for the trivial case where $\gamma = 0$ — *i.e.*, on the surface itself). Note that $0 \leq \alpha \leq 1$ also follows from Eqn. 17.15.

Second, when $\gamma < 0$, the particle travels inside the cylinder, in which case the solution of interest is the positive one, as depicted by the solid line portion of (a) in Fig. 17.4.

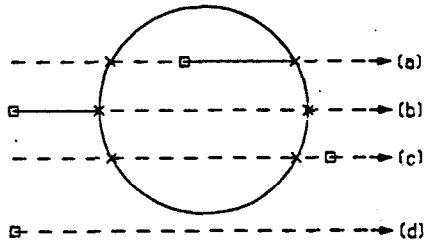

Figure 17.4. Possible trajectories intersecting a cylinder (starting and intersection points indicatd by squares and crosses, respectively).

Finally, when $\gamma > 0$, the particle travels outside the cylinder and several situations present themselves. If $\beta^2 < \alpha\gamma$, there are no real solutions, which represents a particle completely missing the cylinder (*e.g.*, see (d) in Fig. 17.4). The "grazing" solution, corresponding to $\beta^2 = \alpha\gamma$, is also not of particular interest to us. However, when $\beta^2 > \alpha\gamma$, two real solutions exist—either both negative ($\beta > 0$) or both positive ($\beta < 0$), as depicted by (c) and (b), respectively, in Fig. 17.4. Of course, we are only interested in the *smaller* positive solution.

One can easily find solutions for particle trajectories intersecting other conic surfaces (*e.g.*, spheres and cones), including those that have been translated and rotated relative to the defining coordinate system. In the following section, we will show how the above equations are put to use in an auxiliary subprogram that comes with the EGS4 Code System.

17.1.4 The CYLNDR algorithm available in EGS4.

SUBROUTINE CYLNDR (see Fig. 17.5) of the EGS4 Code System provides an example of an algorithm for determining the intersection of a particle trajectory with a cylinderical surface using the equations developed above. As in the case of SUBROUTINE PLANE1, the components of $\vec{X}$ and $\hat{U}$ are available from EGS4 by means of COMMON/STACK/. The cylinder radius is passed in COMMON/CYLDTA/ (for a maximum of 75 cylinders in this example). To make the algorithm more efficient, the user is expected to supply pre-knowledge about whether the current position of the particle is inside (INCY=1) or outside (INCY=0) the cylinder.

A difficulty can arise for a particle traveling very close to the cylinder surface, particulary at a glancing angle. For a given machine precision, the quadratic solutions, together with the way EGS4 uses them, can result in a particle being "stepped" sideways to the surface. Albeit a very small amount, this can cause enough error in the true position of a particle to confuse the user's boundary tracking program (*i.e.*, SUBROUTINE HOWFAR), generally with the result that the program gets caught in an infinite loop. An attempt to resolve such difficulties has been addressed by Stevenson[2], and those portions of the algorithm involving the parameter DELCYL are a direct result of this study. Aside from that, the algorithm in CYLNDR is based precisely on Eqn. 17.13 and Eqn. 17.14.

As a practical matter, the use of a DELCYL-value of 1.0E-4 cm has been found to work satisfactorily in most situations. For very small cylinders, the user may need to reduce the value to 1.0E-5 (or even 1.0E-6). However, for very large cylinders (*e.g.*, $R \approx 10$ cm or larger), a choice of 1.0E-3 may be required in order to avoid the infinite-loop syndrome. The user can experiment with DELCYL and change its value in a dynamic way, as dictated by the particular problem at hand. Alternatively, one can select a small number for DELCYL, such as 1.0E-6, and perform the entire calculation with higher precision. The AUTODBL option associated with FORTRAN compilers on IBM computers provides an easy way to accomplish this feat without having to re-code EGS4.

17.2 GEOMETRY CONSIDERATIONS IN THE EGS4 CODE SYSTEM

17.2.1 The EGS4 User Code concept.

The EGS4 code itself consists of two *user-callable* subroutines, HATCH and SHOWER, which in turn call the other subroutines in the EGS4 code, some of which call two *user-written* subroutines, AUSGAB and HOWFAR. The latter determine the output (scoring) and geometry, respectively. The user communicates with EGS4 by means of various COMMON variables, and is required to write a MAIN *driver* program which, together with AUSGAB and HOWFAR (and any auxiliary subprograms), consititute what is commonly referred to as the EGS4 *User Code* (*e.g.*, see Chapter 12).

In general, any geometry initialization needed by HOWFAR is performed in MAIN. Although the user most typically "hard codes" the various geometry parameters within MAIN itself, there really is no restriction on how this is done. To this end, "card"-input systems similar to those used in MORSE-CG[3] or in FLUKA[4,5] can be implemented, even though this is not a general feature of EGS4. An example of how to adapt the MORSE-CG system to EGS4 is provided by the User Code called UCSAMPCG MORTRAN (file #58 on the EGS4 Distribution Tape).

```
SUBROUTINE CYLNDR(ICYL,INCY,IHIT,TCYL);
"-------------------------------------------------------------------------"
"  Input:  ICYL      Cylinder identification number                       "
"          INCY = 1  Current particle position is INSIDE cylinder        "
"               = 0  Current particle position is OUTSIDE cylinder       "
"  Output: IHIT = 1  Particle trajectory HITS surface                    "
"               = 0  Particle trajectory MISSES surface                 "
"          TCYL      Distance (shortest) to surface (when IHIT=1)       "
"-------------------------------------------------------------------------"
COMMON/STACK/E(40),X(40),Y(40),Z(40),U(40),V(40),W(40),DNEAR(40),
          WT(40),IQ(40),IR(40),NP;
DOUBLE PRECISION E;
COMMON/CYLDTA/CYRAD2(75);
DATA DELCYL/1.0E-4/;   "Close approach parameter for easy getaway"

ALPHA=U(NP)*U(NP) + V(NP)*V(NP);
IF (ALPHA.EQ.0.0) [ IHIT=0; "Parallel to z-axis (indeterminant)" ]
ELSE [
  BETA=X(NP)*U(NP) + Y(NP)*V(NP);
  GAMMA=X(NP)*X(NP) + Y(NP)*Y(NP) - CYRAD2(ICYL);

  "Define some local variables next"
  ACYL=SQRT(ALPHA); BCYL=BETA/ACYL; CCYL=GAMMA; ARGCY=BCYL*BCYL - CCYL;

  IF (ARGCY.LT.0.0) [ IHIT=0; "Imaginary solutions" ]
  ELSE [ "Real solutions (treating machine precision difficulties)"

    IF (ABS(CCYL).LT.DELCYL.AND.INCY.EQ.0.AND.BCYL.GE.0.0)[ IHIT=0; ]
    ELSEIF (ABS(CCYL).LT.DELCYL.AND.INCY.EQ.1.AND.BCYL.LT.0.0)[
      IHIT=1; TCYL=-2.0*BCYL/ACYL;
      ]
    ELSE [
      IF (INCY.EQ.1.AND.CCYL.GE.0.0) [
        IHIT=1; TCYL=DELCYL;
        ]
      ELSEIF (INCY.EQ.0.AND.CCYL.LE.0.0) [
        IHIT=1; TCYL=DELCYL;
        ]
      ELSE [ "Normal hit-or-miss solutions"
        IF (CCYL.LT.0.0) [ "Inside cylinder"
          IHIT=1; TCYL=(-BCYL + SQRT(ARGCY))/ACYL;
          ]
        ELSEIF (BCYL.LT.0.0) [ "Outside (and moving towards) cylinder"
          IHIT=1; TCYL=(-BCYL - SQRT(ARGCY))/ACYL;
          ]
        ELSE [ IHIT=0; "Outside (but moving away from cylinder)" ]
        ]
      ]
    ]
  ]
RETURN;
END;
```

Figure 17.5. Listing of SUBROUTINE CYLNDR.

17.2.2 Specifications for (and an example of) HOWFAR.

On entry to the geometry subprogram HOWFAR, EGS4 has determined that it would like to transport the top particle on its stack (identified by NP) by a straight-line distance USTEP. The state-function parameters of the particle are available to the user via COMMON/STACK/ as described previously. The user controls the transport by setting the variables USTEP, IDISC, IRNEW and, in some instances, DNEAR(NP).

Except for the last variable, which is in COMMON/STACK/, they are available to the user via COMMON/EPCONT/. The various ways in which they may be changed, and the way EGS4 will interpret these changes, is discussed in great detail in the EGS4 User Manual[1], and we will not duplicate that effort here. Instead, we will give a simple HOWFAR example using the PLANE1 and CYLNDR routines discussed above, which should provide the reader with a reasonable idea of how geometries are generally handled in the EGS4 Code System.

Consider a cylindrical target struck by an incident electron beam as shown in Fig. 17.6. The cylinder of rotation about the z-axis is identified by Box 1. There are four regions of interest—the target (region 2) and three vacuum regions upstream, downstream, and surrounding the target. The extent of the target along the z-direction is determined by two end planes, identified by Triangles 1 and 2 that point in the direction of the defining unit normal vectors:

$$\begin{aligned} \texttt{PNORM}(1,1) &= 0.0 \\ \texttt{PNORM}(2,1) &= 0.0 \\ \texttt{PNORM}(3,1) &= 1.0\ , \end{aligned} \tag{17.16}$$

and

$$\begin{aligned} \texttt{PNORM}(1,2) &= 0.0 \\ \texttt{PNORM}(2,2) &= 0.0 \\ \texttt{PNORM}(3,2) &= 1.0\ . \end{aligned} \tag{17.17}$$

The target length, T, is

$$T = \texttt{PCOORD}(3,2) - \texttt{PCOORD}(3,1) \quad \text{(cm)} \tag{17.18}$$

(all the other PCOORD-values are zero), and the radius, R is defined by

$$R^2 = \texttt{CYRAD2}(1) \quad (\text{cm}^2), \tag{17.19}$$

where all the quantities are (generally) defined in MAIN and passed to HOWFAR in COMMON/PLADTA/ and COMMON/CYLDTA/, as we have indicated earlier.

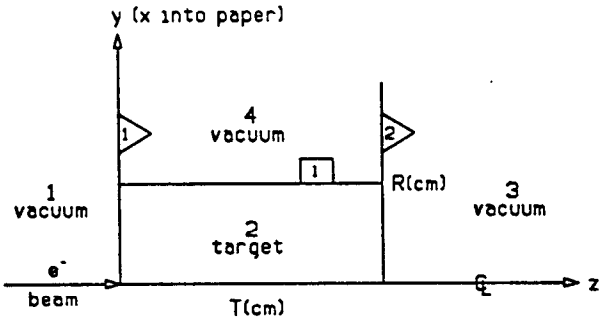

Figure 17.6. Cylinder of rotation about the z-axis bounded by two planes.

With EGS4, the actual transport of electrons and photons is performed in subroutines ELECTR and PHOTON, respectively. In the default version (*i.e.*, no electric or magnetic fields), a simple translation is done; namely,

$$\vec{X}_{new} = \vec{X} + \ell\hat{U} \,, \tag{17.20}$$

where $\vec{X}_{new}$ is the new vector position of the particle after being translated the distance ℓ (*i.e.*, variable USTEP in the code). However, before the transport takes place, HOWFAR is called by one of these subprograms in order to allow the user the opportunity to "interact" with the process–*e.g.*, shorten ℓ, stop a process entirely, etc.

For purpose of illustration, we will assume that the incident electron initially starts out in region 2, and that all particles leaving the target are to be "discarded". Referring to the listing given in Fig. 17.7, the way this is accomplished is to change IDISC=0 (default) to IDISC=1 (*i.e.*, from *no-discard* to *discard*) whenever a particle is somewhere other than in region 2, and to return to the calling program. EGS4 will handle the rest, including calling AUSGAB in order to allow the user to score quantities of interest—*e.g.*, backscattered electrons, forward bremsstrahlung, lateral energy escape, etc.

```
SUBROUTINE HOWFAR;
"------------------------------------------------------------------------"
" Cylinder of rotation about the z-axis bounded by two planes.          "
"------------------------------------------------------------------------"
COMIN/CYLDTA,EPCONT,PLADTA,STACK/; "(See text for explanation)"

IF (IR(NP).NE.2) [ IDISC=1; "Discard particles outside the target" ]

ELSE [ "Track particles within the target"

  CALL CYLNDR(1,1,IHIT,TCYL); "Check the cylinder surface"
  IF (IHIT.EQ.1) [ "Surface is hit---make changes if necessary"
    IF (TCYL.LE.USTEP) [USTEP=TCYL; IRNEW=4;]
    ]

  CALL PLANE1(2,1,IHIT,TPLN); "Check the downstream plane"
  IF (IHIT.EQ.1) [ "Surface is hit---make changes if necessary"
    IF (TPLN.LE.USTEP) [USTEP=TPLN; IRNEW=3;]
    ]
  ELSEIF (IHIT.EQ.0) [ "Heading backwards"
    CALL PLANE1(1,-1,IHIT,TPLN); "To get TPLN-value (IHIT=1, a must)"
    IF (TPLN.LE.USTEP) [USTEP=TPLN; IRNEW=1;] "Make necessary changes"
    ]
  ]

     RETURN;
     END;
```

Figure 17.7. A simple example of how to write SUBROUTINE HOWFAR.

In this example, all of the particle transport takes place in region 2 where it is HOWFAR's job to decide whether or not the current size of ℓ =USTEP is such that $\vec{X}_{new}$ remains within the target boundaries. Specifically, one must determine the distance from the particle's current position to a point of intersection with a boundary, denoted by t in the previous sections involving subroutines CYLNDR and PLANE1. If $\ell < t$, the translation is allowed and one can return to the calling program without further ado.

On the other hand, both USTEP and IRNEW (the new region in which the particle will eventually end up), will have to be changed if $\ell \geq t$.

Since one does not know *a priori* which surface is intersected "first", all surfaces must be checked in order to obtain the smallest t. A little reflection should convince the reader that it does not matter in which order the calls are made, but all surfaces must be checked. The small exception to this statement, of course, involves two parallel planes; namely, if the first plane is "hit", one does not have to bother with the second plane. Furthermore, if it is known beforehand that the radiation prefers (statistically) to strike one plane rather than the other, the user can take advantage of the fact and force a call to the preferred plane first (*e.g.*, the "downbeam" plane).

Together with earlier discussions about PLANE1 and CYLNDR, the listing in Fig. 17.6 demonstrates how the user should construct SUBROUTINE HOWFAR, at least for the example presented. However, the statement

```
COMIN/CYLDTA,EPCONT,PLADTA,STACK/;,
```

still requires some clarification. Without going into detail at this point, suffice it to say that a COMIN statement is a *macro* that is part of the EGS4 Code System, providing the user with a compact way of writing COMMON statements. The EGS4 Code System is written entirely in the Mortran3 language[6] which has a *macro facility* for accomplishing such tasks (additional information on Mortran3 macros is provided in the EGS4 Code System documentation[1]).

17.2.3 Auxiliary geometry subprograms available with EGS4.

A variety of subprograms, designed to aid the user in creating relatively sophisticated User Codes, are available with the EGS4 Code System* . In the previous sections, we have taken a close look at CYLNDR and PLANE1, including the mathematics that forms the basis of them. We will not go into further mathematical detail in this section since the basic idea and methods are the same. Instead, we will simply itemize which subroutines are available, and provide a terse description of their main features and use. Each routine is self-documented by means of commentary contained within the coding.

- PLANE1 — Determines if particle trajectory strikes a planar surface. Returns trajectory distance (TPLN) (see Section 17.1.2).
- CYLNDR — Determines if the particle trajectory strikes a cylindrical surface. Returns trajectory distance (TCYL) (see Section 17.1.4).
- CONE — Determines if the particle trajectory strikes a conical surface. Returns trajectory distance (TCON).
- SPHERE — Determines if the particle trajectory strikes a spherical surface. Returns trajectory distance (TSPH).
- CHGTR — Changes USTEP and IRNEW whenever USTEP is larger than the trajectory distance (TPLN, TCYL, TCON, TSPH).
- FINVAL — Determines the coordinates of the particle trajectory at the point of an intersection with a given surface.

---

* *e.g.*, see file GEOMAUX MORTRAN (#26) on the EGS4 Distribution Tape.

PLAN2P — Determines the intersection point for two parallel planes by calling PLANE1 twice (when necessary), and CHGTR if a plane is hit.

PLAN2X — Determines the intersection point for two crossing planes by calling PLANE1 twice (always), and CHGTR if a plane is hit (PLAN2X is slightly less efficient than PLAN2P).

CYL2 — Similar to PLAN2P, but for concentric cylinders.

CON2 — Similar to PLAN2P, but for concentric cones.

SPH2 — Similar to PLAN2P, but for concentric spheres.

As an example that demonstrates how the writing of SUBROUTINE HOWFAR can be simplified with the aid of the routines listed above, consider the example given in the previous section (see Fig. 17.7). Using SUBROUTINE PLAN2P, eight lines of code involving two calls to PLANE1 can be replaced by a single call as shown in Fig. 17.8 .

```
SUBROUTINE HOWFAR;
"------------------------------------------------------------------------"
" Cylinder of rotation about the z-axis bounded by two planes.          "
"------------------------------------------------------------------------"
COMIN/CYLDTA,EPCONT,PLADTA,STACK/; "(See text for explanation)"

IF (IR(NP).NE.2) [ IDISC=1; "Discard particles outside the target" ]

ELSE [ "Track particles within the target"

  CALL CYLNDR(1,1,IHIT,TCYL); "Check the cylinder surface"
  IF (IHIT.EQ.1) [ CALL CHGTR(TCYL,4); ] "Change if necessary"

  CALL PLAN2P(2,3,1,1,1,-1); " Check the downstream plane first and"
                             " then the upstream one if necessary"
  ]

      RETURN;
      END;
```

Figure 17.8. Simplification of the previous SUBROUTINE HOWFAR listing.

We have also made use of CHGTR, which admittedly is somewhat trivial in the present example and, in fact, may even slow things down due to "overhead" costs involved in calling subprograms. However, we do gain in *modularity* and code *readability* and, as we shall discuss shortly, one can completely eliminate this overhead with the use of *macro equivalents* in place of the conventional CALL statements. Before explaining this, however, we will give another very practical example.

Consider an electromagnetic cascade shower counter made up of alternating slabs of material along the z-axis (*e.g.*, Pb-scintillator layers) and bounded in the x- and y-directions by two pairs of planes to form a right parallelepiped—*i.e.*, a box of slabs. Since we have already pointed out that (almost) all surfaces must be checked to see if they are hit, it becomes apparent that SUBROUTINE HOWFAR consists primarily of three successive PLAN2P calls:

Two x-planes — CALL PLAN2P(IPLNX1,IRGNX1,1,IPLNX2,IRGNX2,1);

Two y-planes — CALL PLAN2P(IPLNY1,IRGNY1,1,IPLNY2,IRGNY2,1);

Many z-planes — `CALL PLAN2P(IPLNZ1,IRGNZ1,1,IPLNZ2,IRGNZ2,-1);`.

The reader should consult the code listing for PLAN2P for an explanation of the calling parameters. The important point in this example is the significant reduction of coding effort as a result of using PLAN2P. With more complicated geometries, this becomes even more apparent, as demonstrated in a recent paper by Nelson and Jenkins[7].

17.2.4 Mortran3 and macro forms of the geometry routines.

The EGS4 Code System relies heavily on *extensions* to the Mortran3 language in the form of a set of macros that reside in the file called EGS4MAC MORTRAN (#20 on the EGS4 Distribution Tape)* . One of the new features in EGS4 is the addition of a set of *geometry macros* that perform the same task as the subroutines listed in the previous section, but in a more efficient way since the sole purpose of each macro is to place subroutine code *directly in-line*. In other words, the overhead of performing an external call is obviated.

We will not go into any detail on geometry macros other than to give a terse example to show how things work. Consider in Fig. 17.8 the statement

```
IF (IHIT.EQ.1) [ CALL CHGTR(TCYL,4); ] ,
```

which performs the same task as the few lines of code in Fig. 17.7. The equivalent *macro template* would look like

```
IF (IHIT.EQ.1) [ $CHGTR(TCYL,4); ] ,
```

which would get operated on by the following *replacement macro*:

```
REPLACE {$CHGTR(#,#);} WITH
  {IF({P1}.LE.USTEP) [USTEP={P1}; IRNEW={P2};]} .
```

The first # assigns TCYL to {P1}, the second # assigns 4 to {P2}. The replacement code is then inserted directly at the location of $CHGTR, such that the resultant code looks like:

```
IF (IHIT.EQ.1) [
  IF (TVAL.LE.USTEP) [USTEP=TCYL; IRNEW=4;]
  ]
```

In other words, we get the same in-line code that we originally started with (see Fig. 17.7)—*i.e.*, we get modularity, readability, and speed.

With this digression behind us, suffice it to say that the geometry macros are only slightly more complicated than the example above. They are used in the same way as

---

* See Chapter 12 for a discussion of Mortran3 macros.

```
SUBROUTINE HOWFAR;
"-------------------------------------------------------------------------"
" Cylinder of rotation about the z-axis bounded by two planes.           "
"-------------------------------------------------------------------------"
COMIN/CYLDTA,EPCONT,PLADTA,STACK/; "(See text for explanation)"

IF (IR(NP).NE.2) [ IDISC=1; "Discard particles outside the target" ]

ELSE [ "Track particles within the target"

  $CYLNDR(1,1,IHIT,TCYL); "Check the cylinder surface"
  IF (IHIT.EQ.1) [ $CHGTR(TCYL,4); "Change if necessary" ]

  $PLAN2P(2,3,1,1,1,-1); " Check the downstream plane first and"
                           " then the upstream one if necessary"
  ]

      RETURN;
      END;
```

Figure 17.9. SUBROUTINE HOWFAR listing using macros instead of CALL statements.

CALL statements are used. Thus, the example that we have been following could have been written as shown in Fig. 17.9 .

17.2.5 Other EGS4-related geometry packages.

A general purpose EGS4 User Code to do Cartesian coordinate dose deposition studies has been designed by Rogers[8]. This User Code, called XYZWRN.MOR, makes use of $PLAN2P and associated macros to construct SUBROUTINE HOWFAR. Rectangular parallel beams of photons or electrons are incident on the x-y surface at an arbitrary angle relative to the z-direction. Every voxel (volume element) can have different materials and/or varying densities (*e.g.*, for use with CT data input). Voxel dimensions are completely variable in all three directions.

A similar Cartesian geometry package has been designed by Stevenson[9] . Although it is not presented with any particular application in mind, it does demonstrate how one uses the $PLANE1 and $PLAN2P (also called $PLANE2) macros to create a rectilinear system of volume elements, using a region numbering system similar to that used in FLUKA[4,5] .

General purpose geometry routines are convenient for many applications, although the ease in not having to write the code can cost CPU time. When the geometry is very regular, as is so often the situation, substantial savings (*e.g.*, 20% to 40%) can be obtained using special purpose coding[8] .

17.3 COMBINATORIAL GEOMETRY

Geometry routines were devised to make defining complex geometries relatively simple for the user. An early, and relatively good, example of such a routine is Combinatorial Geometry, developed by MAGI[10] and subsequently adapted for use in the MORSE[3] Monte Carlo neutron and gamma-ray transport program. It has since been added to FLUKA[4,5], ACCEPT[11], and to the EGS4 Code System in the form of a specifically designed User Code called UCSAMPCG (file #58 on the Distribution Tape).

The combinatorial method of specifying input zones in solid bodies is usually more intuitive and simpler than specifications in terms of boundary surfaces. Combinatorial Geometry describes complex three-dimensional configurations by the use of unions, differences and intersections (OR and AND boolean algebraic equations) of simple geometric bodies. The geometric description subdivides the problem space into zones which are the result of combining one or more of the following simple bodies.

RPP — Right Parallelepiped with sides that are parallel to x, y, and z, the minimum and maximum values of which are specified.

BOX — RPP arbitrarily oriented in space. Three mutually perpendicular vectors are specified.

SPH — Sphere. The vertex and a scaler denoting the radius, R, are specified.

RCC — Right Circular Cylinder. The vertex of the center of the base, the height vector, and a scaler denoting the radius, R, are specified.

REC — Right Elliptical Cylinder. The coordinates of the center of the base ellipse, the height vector, and two vectors in the plane of the base defining the major and minor axes are specified.

TRC — Truncated Right Angle Cone. The vertex of the center of the base, the height vector, and two scalers denoting the radii of the upper and lower bases are specified.

ELL — Ellipsoid. Two vertices denoting the foci, and a scaler, R, denoting the length of the major axis are specified.

WED — Right Angle Wedge*. Three mutually perpendicular vectors are specified.

ARB — Arbitrary Convex Polyhedron (4, 5 or 6 sides). An index to each vertex is assigned, and the x, y and z coordinates for each are given. Each of the six faces are then described by a four-digit number giving the indices of the 4 vertex points in that face. (If there are less than six faces, zeros are entered for the non-existent vertices.) For each face, these indices must be entered in either clockwise or counterclockwise order.

All body types except the right parallelepiped may be oriented arbitrarily with respect to the x, y and z coordinate axes describing the space, but the RPP body type must have sides which are parallel to x, y and z.

A special operator notation involving the symbols (+), (−) and (OR) is used to describe the intersections and unions of these simple bodies. Whenever a body appears in a zone description with a (+) operator, it means that the zone being described is wholly contained within the body, whereas if the body appears in a zone description with a (−) operator, it means that the zone being described is wholly outside the body. If the

* Also denoted RAW.

body appears with an (OR) operator, it means that the zone being described includes all points in the body. Sometimes, a zone may be described in terms of subzones lumped together by (OR) statements. Subzones are formed as intersects, and then the zone is formed by the union of these intersects.

When (OR) operators are used, there are *always* two or more of them, and they refer to all body numbers which follow. That is, all body numbers between (OR)'s, or until the end of the zone cards for that zone, are intersected together before the (OR)'s are performed. The body types (except RPP) are shown in Fig. 17.10, and the right parallelepiped (RPP) is shown in Fig. 17.11.

Table 17.1 gives the input required for these geometrical bodies as required by the Combinatorial Geometry package used either in MORSE or EGS.

Table 17.1. Input Required for Various Bodies in Combinatorial Geometry.

ITYPE	IALP	Real data defining Particular Body						Number of
3-5	7-10	11-20	21-30	21-40	41-50	51-60	61-70	cards needed
BOX	assigned	Vx	Vy	Vz	H1x	H1y	H1z	1 of 2
	by user	H2x	H2y	H2z	H3x	H3y	H3z	2 of 2
RPP	or by	Xmin	Xmax	Ymin	Ymax	Zmin	Zmax	1
SPH	code if	Vx	Vy	Vz	R			1
RCC	left	Vx	Vy	Vz	Hx	Hy	Hz	1 of 2
	blank.	R						2 of 2
REC		Vx	Vy	Vz	Hx	Hy	Hz	1 of 2
		R1x	R1y	R1z	R2x	R2y	R2z	2 of 2
ELL		V1x	V1y	V1z	V2x	V2y	V2z	1 of 2
		R						2 of 2
TRC		Vx	Vy	Vz	Hx	Hy	Hz	1 of 2
		R1	R2					2 of 2
WED or		Vx	Vy	Vz	H1x	H1y	H1z	1 of 2
RAW		H2x	H2y	H2z	H3x	H3y	H3z	2 of 2
ARB		V1x	V1y	V1z	V2x	V2y	V2z	1 of 5
		V3x	V3y	V3z	V4x	V4y	V4z	2 of 5
		V5x	V5y	V5z	V6x	V6y	V6z	3 of 5
		V7x	V7y	V7z	V8x	V8y	V8z	4 of 5
		Face descriptions						5 of 5

Note: $\vec{a}$ = (H1x, H1y,H1z), etc.

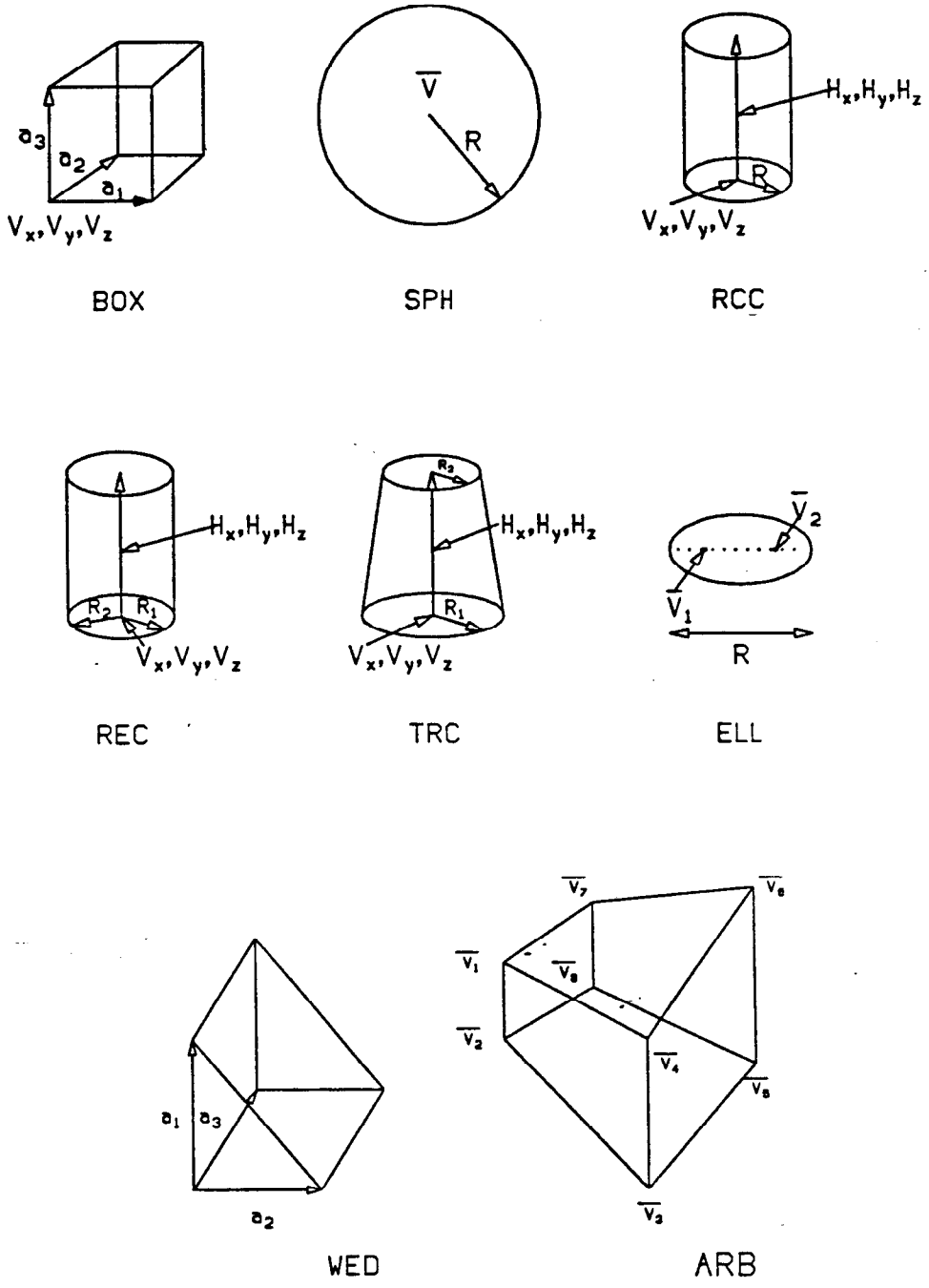

Figure 17.10. Body types and required input dimensions (Combinatorial Geometry).

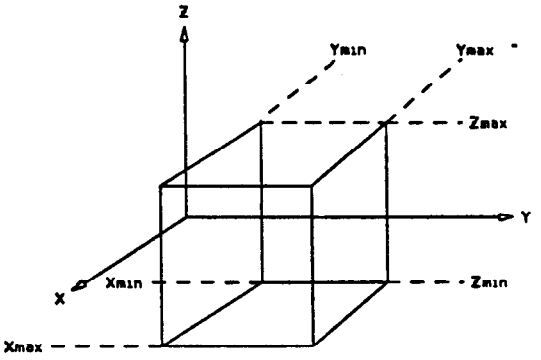

Figure 17.11. Right parallelepiped (RPP) with required dimensions (Combinatorial Geometry).

17.3.1 Constructing bodies using Combinatorial Geometry.

A problem geometry is constructed by

1. Defining the location and orientation of each body required for specifying the input zone.
2. Specifying the input zones as combinations of these bodies.
3. Specifying the volumes of the input zones (if necessary).
4. Specifying the material in each input zone.

As an example, take the intersection of a sphere and a cylinder as shown in Fig. 17.12. The location and orientation of each body (SPH and RCC) would be specified with cards similar to those of Table 17.1. The various zones, described by the input cards, are shown under the different combinations. If the zones can be considered as *common* (*e.g.*, the same material), as in Fig. 17.12(c), the description would be OR +1OR +2. That is, the zone is composed of *either* the sphere (Body 1) or the cylinder (Body 2). If there are to be two zones as shown in Fig. 17.12(b), or Fig. 17.12(d), the zone descriptions would be; (b) Zone 1: +1; Zone 2: +2 −1; and (d) Zone 1: +1 −2; Zone 2: +2. Where three zones are defined as in Fig. 17.12(e), the zones would be described by; Zone 1: +1 −2; Zone 2: +2 −1; Zone 3: +2 +1. Note that in all these examples, each zone is described only once. That is, a particle can be in only one zone. In Fig. 17.12(c), the geometry defines only one zone. But in the others, there are multiple zones, and each zone is defined in only one place. It is important that every spatial point in the geometry be located in one, and only one, zone since otherwise the program will fail (with appropriate error messages).

When constructing a geometry, one should also try to avoid defining bodies which share a common boundary surface since the program might have difficulties determining in which zone a particle is located. A small overlap is preferable. At the same time, the user must make certain that the overlap is contained in only one of the zones. For example, if Fig. 17.13 were composed of two cylinders which butted against each other, they shouldn't share a common surface as in figure17.13(a), but should overlap as in Fig. 17.13(b). The size of the overlap isn't important since it subsequently disappears in the zone description given below Fig. 17.13(b).

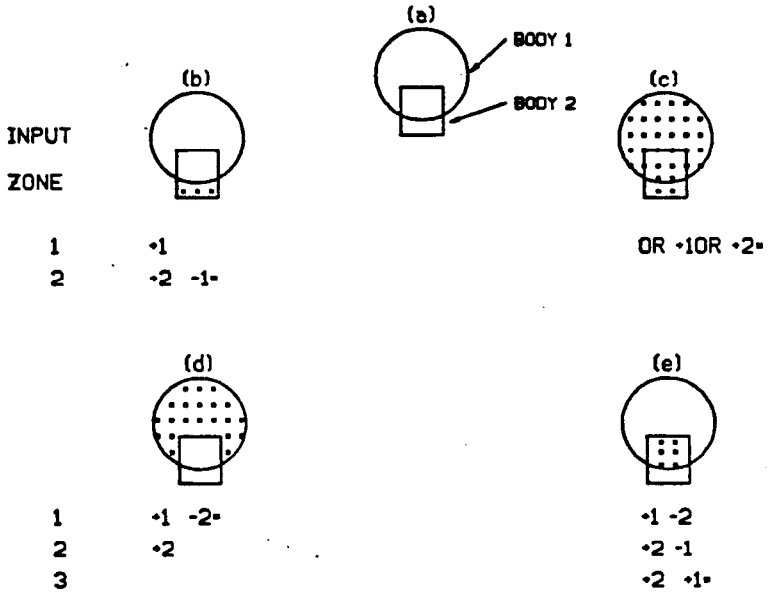

Figure 17.12. Zone descriptions for a sphere (SPH) and a cylinder (RCC) input.

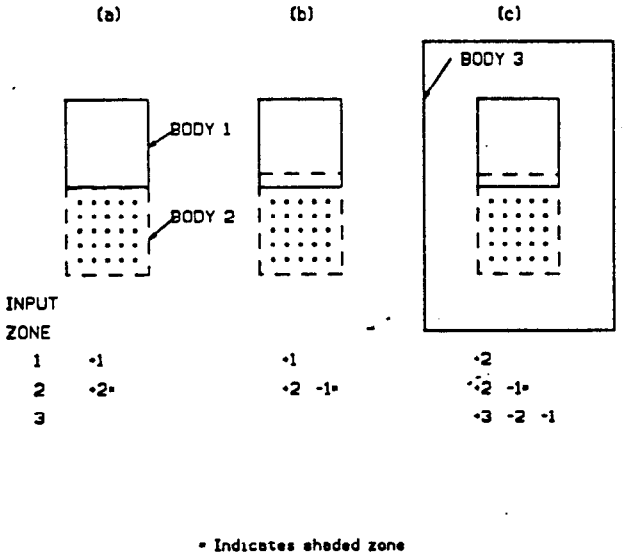

Figure 17.13. Zone description for two cylinders (RCC) which share a common boundary.

The universe for Combinatorial Geometry is defined (limited) by the outermost body which *must* enclose all other bodies within it. In order to turn tracking off (*i.e.*, to have a discard region), this last body is defined as a null region. Fig. 17.13(c) illustrates this last point with the addition of Body 3 surrounding the two cylinders. In

the zone description, Zone 3 would be a null zone, and all particles entering it would immediately be discarded.

17.3.2 An example of a complex MORSE-CG geometry.

As an example of a reasonably complex geometry constructed with Combinatorial Geometry, a typical medical accelerator room (with roof removed for better viewing) is shown in Fig. 17.14 . This geometry was used for transmission as well as room scattering studies for both neutrons and gamma rays. The input code that created the geometry is shown in Fig. 17.15.

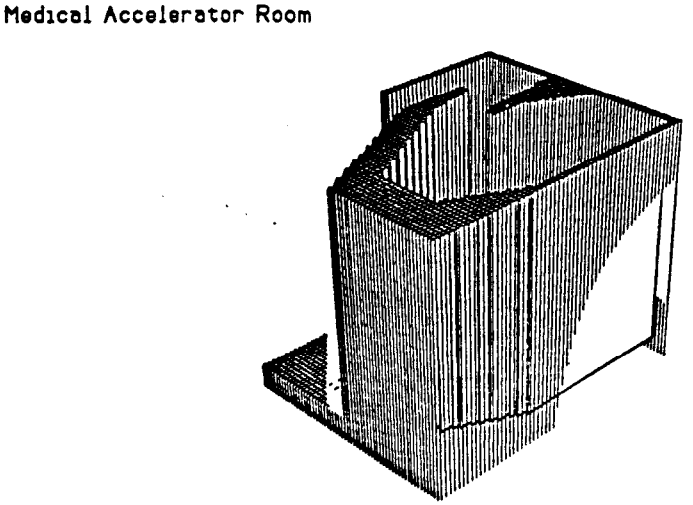

Figure 17.14. A typical hospital medical accelerator room.

A more recent trend in geometry packages has been to define relatively simple shapes using, for example, Combinatorial Geometry (with additional shapes, such as trapeziods, tubes, segments of cylinders and cones, etc), and then to replicate these shapes anywhere in space (*i.e.*, with any orientation and any size) to form complex geometries where repetitive shapes are found, as-is the case in many of the high-energy particle accelerator detectors. However, one should realize that there is a trade-off between ease of use of a geometry package and computer time. The geometries that allow the user to construct intricate geometries most easily may not have the best tracking algorithms.

17.4 GEOMETRY PACKAGES IN ETRAN, ITS AND FLUKA

17.4.1 ETRAN

The ETRAN code[12]*, as it is distributed by the Radiation Shielding Information Center (RSIC) at the Oak Ridge National Laboratory, is capable of treating:

1. A set of homogeneous, semi-infinite slabs (1-D).
2. A set of homogeneous, concentric, right circular cylinders of finite length (3-D).

* Also see Chapters 7 and 8.

```
COMMENT - A MEDICAL ACCELERATOR ROOM INPUT GEOMETRY
 RPP   1   -396.5     518.5    -976.0     411.8    -244.0     290.0
 RPP   2   -366.0     366.0    -945.5     381.3    -132.0     234.0
 ARB   3   -152.5    -738.1    -131.99    -122.0    -738.1    -131.99
           -122.0    -738.1     233.99    -152.5    -738.1     233.99
           -152.5    -655.8    -131.99       0.0    -610.0    -131.99
              0.0    -610.0     233.99    -152.5    -655.8     233.99
        1234.     4158.     8567.     7326.     6512.     3784.
 ARB   4  -121.99    -738.1    -131.99      54.9    -738.1    -131.99
             54.9    -738.1     233.99    -121.99    -738.1     233.99
           365.99    -228.8    -131.99     365.99    -427.0    -131.99
           365.99    -427.0     233.99     365.99    -228.8     233.99
        1234.     4158.     8567.     7326.     6512.     3784.
 WED   5  -365.99    -404.0    -131.99       0.0     -69.0       0.0
           289.7        0.0       0.0       0.0       0.0     365.98
 ARB   6  -106.8    -403.99    -131.99     -76.3    -403.99    -131.99
            -76.3    -403.99     233.99    -106.8    -403.99     233.99
           -183.0    -320.3    -131.99     -76.3    -320.3    -131.99
            -76.3    -320.3     233.99    -183.0    -320.3     233.99
        1234.     4158.     8567.     7326.     6512.     3784.
 WED   7  -365.99    -403.99    -131.99       0.0     266.7       0.0
           259.18       0.0       0.0       0.0       0.0     365.98
 WED   8   365.99    381.29    -131.99       0.0    -228.8       0.0
           -228.7       0.0       0.0       0.0       0.0     365.98
 ARB   9   518.49    -975.99    -131.99    518.49    -274.5    -131.99
           518.49    -274.5     233.99    518.49    -975.99     233.99
           366.01    -975.99    -131.99    366.01    -427.0    -131.99
           366.01    -427.0     233.99    366.01    -975.99     233.99
        1234.     4158.     8567.     7326.     6512.     3784.
 RPP   10     30.5     365.99    -975.99    -945.51    -131.99     233.99
 RPP   11   -400.0      520.0    -980.0      415.0     -250.0      300.0
 END
 CON 10000R  +1     -2     -9     -100R   +50R   +60R   +70R   +80R   +4
       OR    +3
 VAC   OR    +2     -3     -4     -5      -6     -7     -80R   +90R   +10
 NUL         +11    -1
 END
   1    1    1
   1 1000    0
```

Figure 17.15. Combinatorial Geometry input used by MORSE to produce Fig. 17.14.

Many cylinders (or slabs) may be specified in a single run, but they are treated *independently*. That is to say, events that occur within an inner cylinder are useful in obtaining outer cylinder results, but outer cylinder interactions have no effect on the results obtained for the inner cylinders. The same is true for the slab geometry—*i.e.*, an individual Monte Carlo run may involve 10 or 20 thicknesses of the same material, with the thin slab interactions used directly in obtaining the results for the thicker slabs (but not the other way around).

A version of 1-D ETRAN at the National Bureau of Standards[13] also has been developed in order to handle multilayer slabs of different materials. The version that is currently distributed by RSIC, however, is severely limited in its application to a large number of real physical problems, primarily because it lacks a general, versatile, and easy to use geometry package. Overcoming this limitation was the original motivation for the development of the TIGER series, which we shall briefly discuss next.

17.4.2 ITS: The Integrated TIGER Series

TIGER[14], CYLTRAN[15], and ACCEPT[11] are the base codes that constitute what is called the Integrated TIGER Series (ITS)*. They are all based on ETRAN, and differ from one another primarily in their dimensionality and geometric modeling.

The geometry of the TIGER codes is the simplest of the ITS members. It is strictly a one-dimensional code that is essentially the same as the multilayer NBS version of ETRAN:

1. A particle trajectory is described in terms of the z-coordinate of position and the z-direction cosine.
2. Layers are stacked along the positive z-axis beginning at 0.

The CYLTRAN codes employ a fully three-dimensional description of particle trajectories with the material geometry consisting of a right circular cylinder of finite length, the axis of which coincides with the z-axis. CYLTRAN is particularly useful for radiation fields that exhibit cylindrical symmetry, such as electron or photon beams.

The ACCEPT codes provide a method for electron-photon transport through three-dimensional multimaterial geometries described by the Combinatorial Geometry scheme developed by MAGI[10] (see earlier section).

17.4.3 The FLUKA hadronic cascade code.

FLUKA[4,5] is a modular program for computing hadronic and electromagnetic cascades in matter. Designed primarily for use by the high-energy particle physics community, FLUKA may be of interest to a more general audience because:

1. It has been coupled with the EGS4 code in such a manner that electron-photon transport can be done completely within the FLUKA environment, thereby obviating the need to create User Codes (note: the hadron interaction option can be turned off).
2. Several geometry packages are available for general use, and user-defined packages can be implemented in a relatively simple way.
3. Some of the FLUKA geometry packages stand on their own, and provide methods and algorithms that might be of use in other electron-photon codes.

FLUKA provides its own cylindrical, Cartesian, and spherical-conical geometry package. In addition, FLUKA provides access to a modified version of the Combinatorial Geometry package that has been described earlier. All the geometries provide multi-region, multi-medium environments.

FLUKA87 is written in FORTRAN 77, and the user input consists of option cards which are sometimes followed by data cards specific to the option card given. The documentation for FLUKA is rather extensive, representing many man-years of effort by the scientists at CERN-Leipzig-Helsinki who developed the system. In particular, those sections of the User's Guide[4,5] concerned with geometries are well-written, and the card input system is nicely conceived.

---

* Also see Chapter 10.

REFERENCES

1. W. R. Nelson, H. Hirayama and D. W. O. Rogers, "The EGS4 Code System", SLAC-265 (1985).
2. G. R. Stevenson, "A Cylindrical Geometry Package for HOWFAR in the EGS Electron Gamma Shower Program", CERN internal report HS-RP/TM/80-78 (1980); also of related interest: G. R. Stevenson and T. Lund, "A Spherical-Conical Geometry Package for HOWFAR in the EGS Electron Gamma Shower Program", CERN internal report HS-RP/TM/81-30 (1981).
3. E. A. Straker, P. N. Stevens, D. C. Irving and V. R. Cain, "MORSE-CG, General Purpose Monte-Carlo Multigroup Neutron and Gamma-Ray Transport Code With Combinatorial Geometry", Radiation Shielding Information Center (ORNL) report CCC-203 (1976).
4. P. A. Aarnio, A. Fasso, H. J. Moehring, J. Ranft and G. R. Stevenson, "FLUKA86 User's Guide", CERN Divisional report TIS-RP/168 (1986).
5. P. A. Aarnio, J. Lindgren, J. Ranft, A. Fasso and G. R. Stevenson, "Enhancements to the FLUKA86 Program (FLUKA87)", CERN Divisional report TIS-RP/190 (1987).
6. A. J. Cook, "Mortran3 User's Guide", SLAC Computation Research Group technical memorandum CGTM 209 (1983).
7. W. R. Nelson and T. M. Jenkins, "Writing SUBROUTINE HOWFAR for EGS4", Stanford Linear Accelerator report SLAC-TN-87-4 (1987).
8. Private communication with D. W. O. Rogers, National Research Council of Canada (1987).
9. G. R. Stevenson, "A 3-Dimensional Cartesian Geometry Package for HOWFAR in the EGS Electron Gamma Shower Program", CERN Internal report HS-RP/TM/80-60 (1980).
10. W. Guber, J. Nagel, R. Goldstein, P. S. Mettelman and M. H. Kalos, "A Geometric Description Technique Suitable for Computer Analysis of Both the Nuclear and Conventional Vulnerability of Armored Military Vehicles", Mathematical Applications Group, Inc. report MAGI-6701 (1967).
11. J. A. Halbleib, "ACCEPT: A Three-Dimensional Electron/Photon Monte Carlo Transport Code Using Combinatorial Geometry", Sandia National Laboratories report SAND 79-0415 (1979); Nucl. Sci. Eng. **75** (1980) 200.
12. M. J. Berger and S. M. Seltzer, "ETRAN Monte Carlo Code System for Electron and Photon Transport Through Extended Media", Oak Ridge National Laboratory (RSIC) report CCC-107 (1968).
13. Private communication with S. M. Seltzer (August 1987).
14. J. A. Halbleib and W. H. Vandevender, "TIGER, A One-Dimensional Multilayer Electron/Photon Monte Carlo Transport Code", Nucl. Sci. Eng. **57** (1975) 94.
15. J. A. Halbleib and W. H. Vandevender, "CYLTRAN: A Cylindrical- Geometry Multimaterial Electron/Photon Monte Carlo Transport Code", Nucl. Sci. Eng. **61** (1976) 288.