PENGEOM – A general-purpose geometry package for Monte Carlo simulation of radiation transport in material systems defined by quadric surfaces

Julio Almansa[a], Francesc Salvat-Pujol[b], Gloria Díaz-Londoño[c], Artur Carnicer[d], Antonio M. Lallena[e], Francesc Salvat[d,*]

[a] *Servicio de Radiofísica y P.R., Hosp. Univ. Virgen de las Nieves, Avda. de las Fuerzas Armadas 2, 18014 Granada, Spain*
[b] *Institut für Theoretische Physik, Goethe-Universität Frankfurt, Max-von-Laue-Straße 1, 60438 Frankfurt am Main, Germany*
[c] *Departamento de Ciencias Físicas, Universidad de La Frontera, Av. Francisco Salazar, 01145 Temuco, Chile*
[d] *Facultat de Física, Universitat de Barcelona, Diagonal 645, 08028 Barcelona, Spain*
[e] *Departamento de Física Atómica, Molecular y Nuclear, Universidad de Granada, 18071 Granada, Spain*

Abstract

The Fortran subroutine package PENGEOM provides a complete set of tools to handle quadric geometries in Monte Carlo simulations of radiation transport. The material structure where radiation propagates is assumed to consist of homogeneous bodies limited by quadric surfaces. The PENGEOM subroutines (a subset of the PENELOPE code) track particles through the material structure, independently of the details of the physics models adopted to describe the interactions. Although these subroutines are designed for detailed simulations of photon and electron transport, where all individual interactions are simulated sequentially, they can also be used in mixed (class II) schemes for simulating the transport of high-energy charged particles, where the effect of soft interactions is described by the random-hinge method. The definition of the geometry and the details of the tracking algorithm are tailored to optimize simulation speed. The use of fuzzy quadric surfaces minimizes the impact of round-off errors. The provided software includes a Java graphical

[*] Corresponding author.
E-mail address: francesc.salvat@ub.edu

user interface for editing and debugging the geometry definition file and for visualizing the material structure. Images of the structure are generated by using the tracking subroutines and, hence, they describe the geometry actually passed to the simulation code.

Keywords: Constructive quadric geometry; Monte Carlo particle transport; Ray tracing; Geometry visualization

---

PROGRAM SUMMARY

Manuscript Title: PENGEOM – A general-purpose geometry package for Monte Carlo simulation of radiation transport in complex material structures
Authors: Julio Almansa, Francesc Salvat-Pujol, Gloria Díaz-Londoño, Artur Carnicer, Antonio M. Lallena, and Francesc Salvat
Program Title: PENGEOM
Journal Reference:
Catalogue identifier:
Licensing provisions: none
Programming language: Fortran, Java
Computer: PC with Java Runtime Environment installed.
Operating system: Windows, Linux.
RAM: 210 MiB
Supplementary material: Java editor and viewer (PenGeomJar), geometry examples, translator to POV-Ray$^{\mathrm{TM}}$ format, manual.
Keywords: Constructive quadric geometry; Monte Carlo particle transport; Ray tracing; Geometry visualization.
Classification: 21.1 Radiation Physics, 14 Graphics
Nature of problem: The Fortran subroutines perform all geometry operations in Monte Carlo simulations of radiation transport with arbitrary interaction models. They track particles through material systems consisting of homogeneous bodies limited by quadric surfaces. Particles are moved in steps (free flights) of a given length, which is dictated by the simulation program, and are halted when they cross an interface between media of different compositions or when they enter selected bodies.
Solution method: The PENGEOM subroutines are tailored to optimize simulation speed and accuracy. Fast tracking is accomplished by the use of quadric surfaces, which facilitate the calculation of ray intersections, and of modules (connected volumes limited by quadric surfaces) organized in a hierarchical structure. Optimal accuracy is obtained by considering fuzzy surfaces, with the aid of a simple algorithm that keeps control of multiple intersections of a ray and a surface. The Java GUI PenGeomJar provides a geometry toolbox; it allows building and debugging

the geometry definition file, as well as visualizing the resulting geometry in two and three dimensions.
Restrictions: By default PENGEOM can handle systems with up to 5,000 bodies and 10,000 surfaces. These numbers can be increased by editing the Fortran source file.
Unusual features: All geometrical operations are performed internally. The connection between the steering main program and the tracking routines is through a Fortran module, which contains the state variables of the transported particle, and the input-output arguments of the subroutine `step`. Rendering of two- and three-dimensional images is performed by using the PENGEOM subroutines, so that displayed images correspond to the definitions passed to the simulation program.
Additional comments: The Fortran subroutine package PENGEOM is part of the PENELOPE code system. [1]
Running time: The running time much depends on the complexity of the material system. The most complicated example provided, `phantom`, an anthropomorphic phantom, has 264 surfaces and 169 bodies and modules, with different levels of grouping; the largest module contains 51 daughters. The rendering of a 3D image of `phantom` with 1680×1050 pixels takes about 25 seconds (*i.e.*, about $1.5\cdot10^{-5}$ seconds per ray) on an Intel Core I7-3520M CPU, with Windows 7 and subroutines compiled with gfortran.

[1] F. Salvat, J. M. Fernández-Varea, J. Sempau, PENELOPE-*2011: A code System for Monte Carlo Simulation of Electron and Photon Transport*, OECD/NEA Data Bank, Issy-les-Moulineaux, France, 2011. Available from `http://www.nea.fr/lists/penelope.html`.

1. Introduction

Monte Carlo simulation has become a useful tool for solving radiation transport problems. In the simulations, each particle trajectory is generated as a sequence of free flights, each of them ending with an interaction where the particle loses energy, changes its direction of movement, and occasionally produces secondary particles. The length of the free flights and the effects of the interactions are determined by numerical random sampling from probability distributions that are defined by the set of differential cross sections (DCS) of the relevant interaction mechanisms.

Practical simulations involve two different kinds of numerical operations, namely physical (determination of the path length to the next interaction, random sampling of the different interactions) and geometrical (spatial displacements, interface crossings, etc.) Geometry operations are normally performed by dedicated subroutine packages, whose characteristics depend on the type of algorithm used to track particles. The evolution of particles within each homogeneous body is dictated by the physical simulation subroutines, which operate as if particles were moving in an infinite medium with a given composition. The job of the geometry subroutines is to steer the simulation of particle histories in the material system. These subroutines must determine the active medium where a particle is moving, and change it when the particle crosses an interface (*i.e.*, a surface that separates two different media). In the case of material systems with complex geometries, geometrical operations can take a large fraction of the simulation time.

Random particle trajectories can be generated by following different strategies. The so-called *detailed simulation* scheme produces random trajectories by sampling all individual interactions undergone by a particle in chronological succession. Detailed simulation, however, is feasible only when the number of interactions occurring on each particle trajectory is small or moderate (say, up to a few hundred). This is the case for photons, for electrons with energies up to about 50 keV, and for electrons with higher energies (and other charged particles) transported in thin foils. Simulations of charged particles in "thick" media are more time consuming because of the large number of interactions undergone by these particles before being brought to rest (on average, an electron or positron reduces its energy by a few tens of eV at each individual interaction). To cope with this difficulty, charged particles are usually tracked by using condensed simulation schemes (class I schemes in the terminology of Berger [2]) in which each particle trajectory is decomposed into a number of steps (either of fixed or random lengths), and the global effect of all the interactions that occur along each step is described *approximately* by using multiple scattering theories. Because these theories apply to homogeneous infinite media, a limitation of class I schemes occurs when a particle is close to an interface: the step length must be kept smaller than the distance to the nearest interface, to prevent the particle from entering the next medium. Therefore, in class I simulations the geometry subroutines must keep control of the proximity of interfaces. Class II schemes, also called mixed schemes, combine detailed simulation of hard events (*i.e.*, interactions involving energy transfers and polar scattering angles larger than prescribed cutoffs) with condensed simulation of the effect of all soft interactions (with sub-cutoff energy losses and angular deflections) that take place between each pair of successive hard events. Evidently, class II schemes are more accurate than purely condensed simulation because hard events are simulated exactly from the corresponding differential cross sections.

Most general-purpose Monte Carlo codes for high-energy radiation (*e.g.*, ETRAN [3; 4; 5], ITS3 [6], EGS4 [7], GEANT3 [8], EGSnrc [9], MCNP [10], GEANT4 [11; 12], FLUKA [13], EGS5 [14] MCNP6 [15]) utilize detailed simulation for photons, while charged particles are simulated by means of a combination of class I and class II schemes. These codes normally incorporate combinatorial geometry packages (see, *e.g.*, Ref. [16]) which, to ensure proper treatment of interface crossings by charged particles, should provide the distance from the particle's position to the nearest interface.

The electron-photon code PENELOPE [17; 1], as well as an associated code for proton transport [18], makes systematic use of class II schemes for all interactions of charged particles. The accumulated energy loss and angular deflection caused by all soft interactions that occur along a trajectory step are simulated as if they were caused by a single artificial interaction (a hinge), which occurs at a random position within the step. This *random-hinge method* [19] was initially designed to allow the code to operate as in detailed simulations, *i.e.*, the transported particle is moved in straight steps, and the energy and direction of movement change only through discrete interactions (hard interactions and hinges). Kawrakow and Bielajew [20] and Bielajew and Salvat [21] have shown that the random-hinge method provides a quite accurate description of spatial displacements in class I simulations, giving results close to those from detailed simulation. The method works even better for class II schemes, where it only has to account for the effect of soft interactions. A modification of the random hinge method, the so-called dual random hinge method, is employed in EGS5 [14].

Aside from the general-purpose codes mentioned above, many Monte Carlo programs have been developed for specific applications in microdosimetry (see, *e.g.*, Ref. [22], and references therein), x-ray emission and electron-probe microanalysis [23; 24], electron microscopy [25], surface electron spectroscopies [26; 27; 28], and others. In addition, Monte Carlo simulation has been used to assess the reliability of theoretical interaction models through comparisons with measurements under multiple-scattering conditions (see, *e.g.*, [29; 30]). Most of these codes are used only for simple geometries, typically multilayered structures, where geometry operations can be handled easily. Generally, they are not utilized for more complex material structures because of the lack of flexible geometry tools. An exception is the code of Ritchie [24], which implements combinatorial geometry.

In the present article we describe the Fortran subroutine package PENGEOM and complementary tools for Monte Carlo simulation of particle transport in complex geometries. The tracking subroutines are the same as those employed in PENELOPE. These subroutines are both robust and flexible enough to represent quite complex quadric geometries, and they are tailored to optimize simulation speed. They can be utilized in detailed and class II simulations. Note that to determine interface crossings we only need to calculate intersections of particle rays with interfaces, which is much easier than computing the distance of the particle to the nearest interface as required by class I schemes (see, *e.g.*, Ref. [31]). Furthermore, with detailed and class II schemes, physical and geometrical operations are effectively decoupled, so that the PENGEOM subroutines can be used with arbitrary interaction models.

Except for trivial cases, the correctness of the geometry definition is difficult to verify and, moreover, three-dimensional structures with inter-penetrating bodies cannot be easily visualized. We present a Java application named PenGeomJar, a graphical user interface (GUI) that eases the definition of the geometry, allows direct debugging of the geometry definition file, and displays two- and three-dimensional images of the geometry on the computer screen. As the images are generated by PENGEOM, which is run in the background, PenGeomJar is a helpful tool for checking the correctness of the geometry definition.

The present article is organized as follows. Section 2 is devoted to relevant features of quadric surfaces and of the algorithm adopted for ray tracing with fuzzy quadric surfaces. Section 3 describes the general strategy adopted for describing complex material systems, and the use of a genealogical tree of modules for optimization. The geometry is defined by means of a formatted text file; the contents and format of this file are described in Section 4. Section 5 describes the structure and operation of the PENGEOM subroutines, and the use of impact detectors to facilitate the control of particle histories from the steering main program. The Java editor/viewer PenGeomJar is presented in Section 6. Finally, Section 7 describes the distribution package, which contains Fortran source files, the Java application `pengeom.jar`, examples of geometry files, and a manual (file `pengeom-manual.pdf`). The manual provides further information on the numerical algorithm and methods, and on the operation and options of the GUI.

2. Quadric surfaces and ray tracing

The PENGEOM subroutines operate with arbitrary units of length. The only assumption is that the input step lengths and the geometry parameters are expressed in the same units. All the geometry elements, as well as the position $\mathbf{r}$ and the direction of movement $\hat{\mathbf{d}}$ of particles, are referred to the laboratory coordinate system, a right-handed Cartesian reference frame which is defined by the position of its origin of coordinates and the unit vectors $\hat{\mathbf{x}} = (1, 0, 0)$, $\hat{\mathbf{y}} = (0, 1, 0)$, and $\hat{\mathbf{z}} = (0, 0, 1)$ along the directions of the frame axes.

Surfaces can be defined in implicit form, *i.e.*, by means of an equation of the type $F(\mathbf{r}) = 0$, where the function $F(\mathbf{r})$ is assumed to be continuous and differentiable. A surface divides the space into two exclusive regions that are identified by the sign of $F(\mathbf{r})$, the *surface side pointer*, SP. A point with coordinates $\mathbf{r}$ is said to be inside the surface if $F(\mathbf{r}) < 0$ (SP $= -1$), and outside it if $F(\mathbf{r}) > 0$ (SP $= +1$). The surface itself [*i.e.*, the set of points such that $F(\mathbf{r}) = 0$] is the boundary of the two regions. It is worth noting that the equation $-F(\mathbf{r}) = 0$ defines the same surface, but with the inside and the outside interchanged. Consequently, one must be careful with the global sign of the surface function $F(\mathbf{r})$.

The material systems considered in PENGEOM consist of a number of homogeneous bodies B_k determined by their limiting surfaces S_i and compositions (materials). Each surface S_i is specified by giving its equation $F_i(\mathbf{r}) = 0$. The volume of an elementary body can be defined by giving the side pointers SP$_i$ of all the surfaces S_i that limit that volume at an arbitrary point within it. However, using only limiting surfaces may not be convenient for defining complex bodies. In PENGEOM we adopt a more expedient method and consider that bodies are defined in "ascending" exclusive order so that previously defined bodies effectively limit the new ones.

The "location" of a particle is specified by giving its position coordinates $\mathbf{r} = (x, y, z)$ *and* the label k of the body B_k where it is moving. Given the initial position of a particle, $\mathbf{r}_0$, in order to determine the body that contains it, we should calculate the SPs of all limiting surfaces and explore the set of bodies in ascending order to find the (first) one with the right SPs. For complex systems, with a large number of limiting surfaces, this blind search may be quite lengthy. The grouping of bodies into modules (see Section 3) serves to reduce the number of surfaces that need to be considered to locate a point.

As indicated above, the trajectory of a particle is simulated as a sequence of connected straight free flights, each of which ends with an interaction of the particle or with a hinge. When the particle reaches an interface (*i.e.*, a surface limiting two adjacent volumes of different compositions), the simulation has to be halted and restarted with the interaction cross sections of the medium beyond the interface. The most basic geometry operation, which is to be performed millions of times in the course of a simulation run, is the calculation of intersections of particle track segments with limiting surfaces. Let us assume that a particle starts a free flight of length s_0 from a point $\mathbf{r}_0$ in body B_k moving in the direction $\hat{\mathbf{d}}$. We wish to determine whether the track segment intersects any of the surfaces $F(\mathbf{r}) = 0$ that limit that body. The intersections of the ray $\mathbf{r}_0 + s\hat{\mathbf{d}}$ with a surface occur at distances s from $\mathbf{r}_0$ that are solutions of the following "master" equation

$$f(s) \equiv F(\mathbf{r}_0 + s\hat{\mathbf{d}}) = 0, \tag{1}$$

where $f(s)$ is the value of the surface function along the ray. In the course of a free flight of length s_0 the particle will cross the surface only if this equation has a root s such that $0 < s \leq s_0$. Notice that we need to consider only intersections ahead of the ray ($s > 0$).

To simplify the calculation of surface crossings it is convenient to use surfaces expressed by simple analytical functions such that the master equation (1) can be solved analytically. In the PENGEOM subroutines, all limiting surfaces are assumed to be real quadrics defined by the implicit equation

$$\begin{aligned} F(\mathbf{r}) &= A_{\rm xx}x^2 + A_{\rm xy}xy + A_{\rm xz}xz + A_{\rm yy}y^2 + A_{\rm yz}yz + A_{\rm zz}z^2 \\ &\quad + A_{\rm x}x + A_{\rm y}y + A_{\rm z}z + A_0 = 0, \end{aligned} \tag{2}$$

which includes planes, spheres, cylinders, cones, ellipsoids, paraboloids, hyperboloids, etc. It is useful to express the generic quadric equation (2) in matrix form,

$$F(\mathbf{r}) = \mathbf{r}^{\rm T}\mathcal{A}\,\mathbf{r} + \mathbf{A}^{\rm T}\mathbf{r} + A_0 = 0, \tag{3}$$

where

$$\mathcal{A} \equiv \begin{pmatrix} A_{\rm xx} & \frac{1}{2}A_{\rm xy} & \frac{1}{2}A_{\rm xz} \\ \frac{1}{2}A_{\rm xy} & A_{\rm yy} & \frac{1}{2}A_{\rm yz} \\ \frac{1}{2}A_{\rm xz} & \frac{1}{2}A_{\rm yz} & A_{\rm zz} \end{pmatrix} \tag{4}$$

is a symmetric matrix. Here vectors such as $\mathbf{r}$ and $\mathbf{A} \equiv (A_\mathrm{x}, A_\mathrm{y}, A_\mathrm{z})$ are considered as one-column matrices. The gradient of the quadric surface function is the vector

$$\nabla F(\mathbf{r}) = 2\mathcal{A}\mathbf{r} + \mathbf{A}\,. \tag{5}$$

The advantage of using quadric surfaces is that the master equation (1) is quadratic,

$$f(s) = as^2 + bs + c = 0 \tag{6}$$

with

$$a = \hat{\mathbf{d}}^\mathrm{T}\mathcal{A}\hat{\mathbf{d}}\,, \qquad b = 2\hat{\mathbf{d}}^\mathrm{T}\mathcal{A}\mathbf{r}_0 + \hat{\mathbf{d}}^\mathrm{T}\mathbf{A} = \hat{\mathbf{d}}^\mathrm{T}\nabla F(\mathbf{r_0})\,, \qquad c = F(\mathbf{r_0})\,, \tag{7}$$

and its roots are

$$s = \frac{-b \pm \sqrt{\Delta}}{2a} \qquad \text{with} \qquad \Delta \equiv b^2 - 4ac. \tag{8}$$

If the discriminant Δ is positive, there are two real roots and the ray intersects the surface twice; if $\Delta = 0$, there is a real root of multiplicity two and the ray grazes the surface; finally, if Δ is negative, there are no real roots and the ray misses the surface. When $\mathcal{A} = 0$, the surface is a plane and there is only one root, $s = -c/b$.

It is worth mentioning that surfaces defined by cubic or four-degree polynomials in x, y and z also allow the analytical calculation of ray intersections, because the associated function $f(s)$ is a cubic or fourth-degree polynomial in s. However, the calculation is more complicated than for quadric surfaces. Since lengthy geometrical operations may severely impair the efficiency of Monte Carlo simulation, only quadric surfaces are considered in PENGEOM.

To facilitate the definition of the geometry, PENGEOM allows the specification of a surface by giving its "shape" (reduced form) and a set of transformation parameters. The reduced form of a real quadric is given by the expression

$$F_\mathrm{r}(\mathbf{r}) = I_1x^2 + I_2y^2 + I_3z^2 + I_4z + I_5 = 0, \tag{9}$$

where the coefficients (indices) I_1 to I_5 can only take the values -1, 0, or 1. There are 10 possible reduced forms, which define real quadric surfaces of various shapes with "standard" position and orientation. Any real quadric surface can be obtained from its reduced form by means of the following sequence of transformations:

• A scaling along the directions of the axes, defined by the scaling factors X-SCALE, Y-SCALE and Z-SCALE.
• A rotation, defined by the Euler angles OMEGA, THETA, and PHI (see, *e.g.*, [32]). This rotation is the result of a sequence of three right-handed rotations about the coordinate axes: first a rotation of angle OMEGA about the z-axis, followed by a rotation of angle THETA about the y-axis and, finally, a rotation of angle PHI about the z-axis.
• A translation, defined by the components of the displacement vector (X-SHIFT, Y-SHIFT, Z-SHIFT).
Note that these are active transformations (they transform the surface, leaving the coordinate axes unaltered) and are applied in the quoted order.

2.1. **Fuzzy quadric surfaces**

Even with quadric surfaces, we can find numerical ambiguities when locating a particle that is very close to a surface: because of the limited accuracy of floating-point numbers in the computer, we may be unable to assert with confidence whether the particle is inside or outside the surface. Indeed, this kind of ambiguity occurs whenever a particle crosses an interface; due to round-off errors, the values of the surface function at the calculated intersection point may have either sign, although in this case we do know that the particle has just passed the surface. To get rid of numerical ambiguities, all surfaces are considered fuzzy, that is, a surface swells or shrinks very slightly when the particle crosses it so as to ensure that the particle is on the correct side of the surface. Of course, surfaces are not altered; instead, we define the SP of a given surface at the particle position $\mathbf{r}_0$ by considering the relative motion of the particle with respect to the surface.

For quadric surfaces, side ambiguities can be readily solved by considering only the values of the parameters of the master equation (6). The coefficient c is the value of the surface function at $\mathbf{r}_0$ and, in principle, its sign gives the side pointer: SP $= \operatorname{sign}(c)$, where $\operatorname{sign}(c) = +1$ if $c \geq 0$, and $= -1$ if $c < 0$. Ambiguities may occur only when $|c|$ is smaller than a certain small value ϵ, the “fuzziness” level, which will be specified below. The coefficient b is the projection of the gradient of $F(\mathbf{r})$ at $\mathbf{r}_0$ along the direction of flight, *i.e.*, the directional derivative of $F(\mathbf{r})$. Hence, if $b > 0$ the particle “leaves” the surface (moving from inside to outside). Conversely, if $b < 0$ the particle “enters” the surface (moves from outside to inside). The surface is made fuzzy by simply defining the SP at an ambiguous point $\mathbf{r}_0$ (*i.e.*, such that $|c| = |F(\mathbf{r}_0)| < \epsilon$) to be the same as the SP at a position slightly advanced along the ray, where the sign of $F(\mathbf{r})$ is unambiguous. That is, we set SP $= -1$ (inside) if $b < 0$ and SP $= +1$ (outside) if $b > 0$. Summarizing, the side pointer of a fuzzy quadric surface at the point $\mathbf{r}_0$, ambiguous or not, is given by the following simple prescription,

```
if $|c| > \epsilon$, then
    SP=sign(c)
else
    SP=sign(b)
end if
```
(10)

To move a particle from its current position $\mathbf{r}_0$ in body B_k, first the length s_0 of the free flight to the next interaction is sampled by using the cross sections of the material in the current body. The particle then "flies" to a new position, which will be either the end of the step $(\mathbf{r}_0 + s_0\hat{\mathbf{d}})$ or the intersection of the ray with one of the surfaces that limit the current body, whichever occurs first. That is, we must calculate the distances s to the intersections with the limiting surfaces, by solving the corresponding master equations (6), and compare them with the free-flight length s_0. Incidentally, we can use the SPs defined by the calculated c values to verify that the particle effectively is in the assumed body and apply appropriate corrections when this is not the case.

After an interface, defined by the Eq. $F(\mathbf{r}) = 0$, is crossed, the particle is placed at a certain point $\mathbf{r}'$ and the tracking subroutine has to identify the new body by computing the SPs of its limiting surfaces at $\mathbf{r}'$. Although the particle has just crossed one of these surfaces, because of round-off errors, its numerical position coordinates may be at either side of the mathematical surface, and even at a distance such that $|F(\mathbf{r}')| > \epsilon$. When $\mathbf{r}'$ is ambiguous with respect to the surface (*i.e.*, when $|F(\mathbf{r}')| < \epsilon$), the algorithm (10) gives the correct SP. The difficulty arises when $\mathbf{r}'$ is not ambiguous, in which case the numerical position may be either before the mathematical surface (undershot) or beyond it (overshot). In the case of overshooting, the value of $F(\mathbf{r}')$ has the correct sign, and the program will give the correct SP. On the contrary, when there is an undershot, the wrong SP will be assigned. While a slight error in the position coordinates is tolerable, altering the SP implies changing the body and material where the particle moves. With mathematical surfaces, wrong SPs have to be corrected by keeping track of the surfaces that are crossed by the particle, which complicates the tracking algorithm. An advantage of using fuzzy surfaces is that, in the case of an

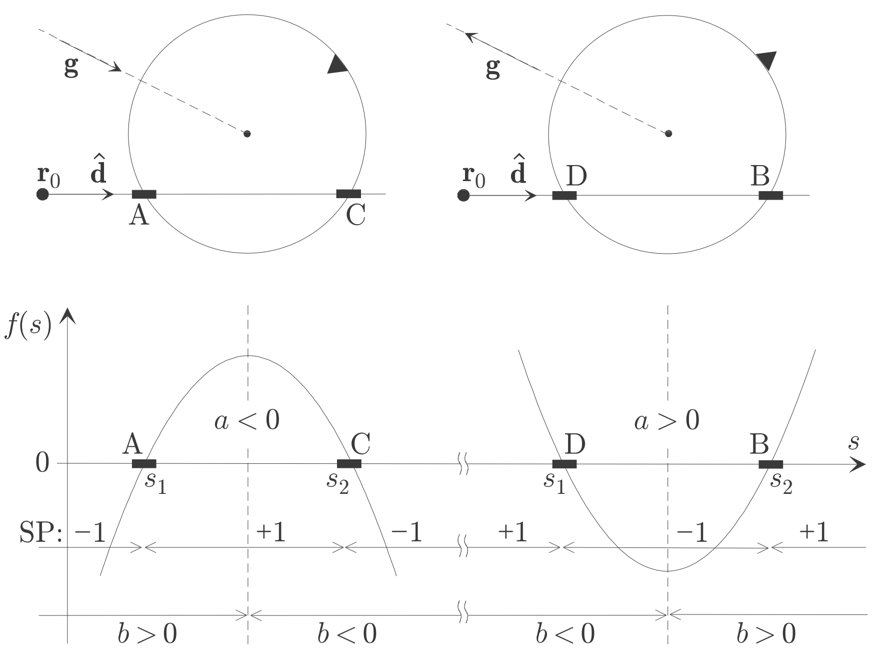


Figure 1: Top: Intersections of rays $\mathbf{r}_0 + s\hat{\mathbf{d}}$ with two quadric surfaces, which for simplicity are represented as spheres; the solid triangles indicate the outside of the surfaces (SP $= +1$). The two surfaces differ only in their orientation (*i.e.*, their defining functions differ by a global negative factor) and, consequently, their gradient vectors $\mathbf{g} = \nabla F(\mathbf{r})$ are in opposite directions. The solid blocks (labelled A, B, C and D) indicate the ray segments (not to scale) where the SP of the surface may be ambiguous, *i.e.*, such that $|F(\mathbf{r})| < \epsilon$. Bottom: Ray intersections described in terms of the master function, $f(s) = as^2 + bs + c$, Eq. (6). The path length s increases to the right. The roots s_1 and s_2 of the equation $f(s) = 0$ are the distances at which the ray intersects the surface, sorted in increasing order ($s_1 < s_2$). The signs of the coefficients a and b depend on the initial position $\mathbf{r}_0$ of the particle in the ray, as indicated.

undershot, the program usually corrects itself because one additional jump of negligible length places the particle at an ambiguous position where the correct SP will be assigned.

As indicated above, the tracking subroutine must determine the distances s from the initial position $\mathbf{r}_0$ of the particle to the intersections of the ray $\mathbf{r}_0 + s\hat{\mathbf{d}}$ with the limiting surfaces of the current body. Non-trivial situations can arise when $\mathbf{r}_0$ is ambiguous with respect to the surface, as it may occur just after a surface crossing, or when the ray intersects the surface twice and the intersections are very close to each other (*e.g.*, when the ray is almost tangent to the surface or when the ray crosses a cone near its tip). In such situations, the tracking subroutine has to evaluate the SP of the surface and also discriminate whether the surface is going to be crossed again or not. If the surface is a plane ($a = 0$), there is only one intersection; the SP is determined by the signs of c and b, and after updating it we can proceed as if the particle had effectively passed the surface. For non-planar quadric surfaces, we need to consider only the case in which the ray does cross the surface (*i.e.*, when the discriminant Δ is strictly positive). We can therefore assume that Eq. (6) has two different roots, s_1 and s_2, sorted in increasing order ($s_1 < s_2$). The general situation is sketched in Fig. 1, which shows a ray crossing a quadric surface and the corresponding "master" function, $f(s) = as^2 + bs + c$. Because the master functions for two $\mathbf{r}_0$ points *in the same ray* differ only by a global displacement in s, the graph of this function is a characteristic of the ray. However, the values of the coefficients a, b, and c do depend on the initial position $\mathbf{r}_0$ in the ray. The lower plot in Fig. (1) shows the signs of a and b corresponding to different initial positions along the ray. We note that for an ambiguous point $|c| = |F(\mathbf{r}_0)| < \epsilon$ and either s_1 or s_2 is close to zero. When s_1 is negative and $s_2 \sim 0$, the ray has just crossed the surface and there are no intersections ahead. When $s_1 \sim 0$ and s_2 is positive, the ray crosses the surface again at s_2. Unfortunately, round-off errors may still lead to ambiguities when both $|s_1|$ and $|s_2|$ are small. Therefore, to reveal the existence of a second intersection ahead, it is preferable to consider some global property of the master function $f(s)$ that is less sensitive to round-off errors. Figure 1 shows that we may have four different situations, which are characterized by the signs of b and a:

A) $b > 0$ and $a < 0$: SP $= +1$, second crossing at s_2.
B) $b \geq 0$ and $a > 0$: SP $= +1$, no more crossings.
C) $b \leq 0$ and $a < 0$: SP $= -1$, no more crossings.
D) $b < 0$ and $a > 0$: SP $= -1$, second crossing at s_2.

When the product ab is negative (cases A and D), $s_1 \sim 0$ and the second root is positive, that is, the ray does intersect the surface again at a certain distance from the initial point $\mathbf{r}_0$, even if the numerical value of s_2 turns out to be negative due to round-off errors. In the latter case, we can simply consider that the intersection is at $s_2 = 0$ because the surface is fuzzy. On the other hand, when $ab > 0$ (cases B and C with $b \neq 0$), $s_2 \sim 0$ and the first root is negative so that there are no intersections beyond $\mathbf{r}_0$. Summarizing, SP ambiguities are resolved by considering the sign of b [*i.e.*, of the derivative of $f(s)$ at $s = 0$], while the existence of a second crossing ahead is readily recognized from the sign of a.

To complete the description of the tracking algorithm, we only have to specify the value of the fuzziness level ϵ, so as to make sure that all points $\mathbf{r}$ such that $|F(\mathbf{r})| > \epsilon$ are assigned the correct SP. Let δ denote the distance that the fuzzy surface will swell or shrink along the direction of the ray. When the surface is not a plane, δ should be much smaller than the distance between the two crossings of the ray, $s_2 - s_1 = \Delta^{1/2}/|a|$. Recalling that the slope of the master function at the intersections is $f'(s_i) = 2as_i + b = \pm\Delta^{1/2}$, we set $\delta = 10^{-12}(s_2 - s_1)$ and

$$\epsilon = f'(s_i)\delta = 10^{-12}\frac{\Delta}{|a|}\,. \tag{11a}$$

In the case of planes ($a = 0$), the fuzziness level is assumed to be a constant, *i.e.*,

$$\epsilon \equiv 10^{-12}\,. \tag{11b}$$

so that the interval of ambiguous points has a half-length $\delta = \epsilon/|f'(s_i)| = 10^{-12}/|b|$, which increases when the angle between the ray and the normal to the plane increases. Note that when $b = 0$ (and $a = 0$) the ray is parallel to the plane and does not intersect it. Numerical experiments, using double-precision arithmetic, confirm that the prescriptions (11) do work well. The precise value of the numerical constant 10^{-12} is not critical; when it is increased, say by a factor of 100, the ambiguity interval widens by approximately that factor (*i.e.*, surfaces become fuzzier), but we have not observed any harmful consequences.

The tracking algorithm is described in Table 1. This algorithm is robust, in the sense that it consistently assigns a SP to ambiguous points $\mathbf{r}_0$ and determines the distance to the next crossing, if there is one. Note, however, that two fuzzy surfaces may not be correctly resolved if they are too close to each other. As a measure of the accuracy of the algorithm we may use the resolution, defined as the minimal distance between two surfaces that are neatly resolved. The resolution is determined by the fuzziness level and by the accumulated numerical round-off errors from the coefficients of the master equation and from the calculation of distances to interfaces. In general, the resolution worsens when the distance to the origin of coordinates increases. That is, a small geometrical structure, which is correctly resolved when placed near the origin, may become distorted or invisible when it is translated to a

Table 1: Algorithm for computing the intersections of a ray, $\mathbf{r}_0 + s\hat{\mathbf{d}}$, with a fuzzy quadric surface. It solves the master equation $f(s) = as^2 + bs + c = 0$ and gives the number n of intersections ahead of the ray. If there are any, the algorithm also provides the corresponding distances s_k ($k = 1$ if $n = 1$; $k = 1, 2$ if $n = 2$) sorted in increasing order. Planes are treated separately to avoid unnecessary calculations.

```
$n = 0$
if $|a| < 10^{-36}$, then   [the surface is a plane]
    if $|b| > 0$, then
        $\epsilon = 10^{-12}$
        if $|c| < \epsilon$, then   [the point $\mathbf{r}_0$ is ambiguous]
            $\mathrm{SP} = \mathrm{sign}(b)$
        else
            $\mathrm{SP} = \mathrm{sign}(c)$
            $t = -c/b$
            if $t > 0$,   set $n = 1$,   $s_1 = t$
        end if
    else   [ray parallel to the plane]
        $\mathrm{SP} = \mathrm{sign}(c)$
    end if
else   [the surface is not a plane]
    $\Delta = b^2 - 4ac$,   $\epsilon = 10^{-12}\Delta/|a|$
    if $|c| < \epsilon$, then   [the point $\mathbf{r}_0$ is ambiguous]
        ambig $= 1$,   $\mathrm{SP} = \mathrm{sign}(b)$
    else
        ambig $= 0$,   $\mathrm{SP} = \mathrm{sign}(c)$
    end if
    if $\Delta < 10^{-36}$, exit   [no "true" intersections]
    $t_1 = \frac{-b}{2a} - \frac{\sqrt{\Delta}}{2|a|}$ ,   $t_2 = \frac{-b}{2a} + \frac{\sqrt{\Delta}}{2|a|}$
    if ambig $= 0$, then
        if $t_1 > 0$,   set $n = 1$,   $s_1 = t_1$
        if $t_2 > 0$,   set $n = n + 1$,   $s_n = t_2$
    else
        if $ab < 0$,   set $n = 1$,   $s_1 = \max\{t_2, 0\}$
    end if
end if
```

further position. With the adopted value of the fuzziness parameter, 10^{-12}, our algorithm is capable of resolving a sphere of unit radius located at a distance of 10^7 length units from the origin.

3. Modules and genealogical tree

During simulation, when a particle attempts to fly a given distance s_0 (step length) in a direction $\hat{\mathbf{d}}$ from a given position $\mathbf{r}_0$ within a body B_0, we need to check whether the particle leaves the body in the course of its flight and, when this occurs, we should halt the particle just after leaving B_0 and resume the simulation with the cross sections of the new material. This requires determining the intersections of the particle ray $\mathbf{r}_0 + s\hat{\mathbf{d}}$ $(0 < s \leq s_0)$ with *all* the surfaces that limit the body B_0 (including those that define any other bodies that limit B_0), and checking whether the final position $\mathbf{r}_0 + s\hat{\mathbf{d}}$ remains within B_0 or not. Because bodies may be concave, we must consider the intersections with all the limiting surfaces even when the final position of the particle lies within B_0. Furthermore, when the particle leaves the initial body, say after travelling a distance s' $(< s_0)$, we need to locate the point $\mathbf{r}' = \mathbf{r}_0 + s'\hat{\mathbf{d}}$.

The straightforward method to locate a point would be to compute the SPs for *all* surfaces and, then, explore the bodies in ascending order looking for the first one that matches the given SPs. This procedure is robust and easy to program, but it becomes too slow for complex geometries. We can speed it up by simply disregarding those elements of the geometry that cannot be reached in a single step (*e.g.*, bodies that are “screened” by other bodies or too far apart). Unfortunately, as a body can be limited by all the other bodies that have been defined previously, the algorithm can be improved only at the expense of providing it with additional information. We adopt a simple strategy that consists of grouping different bodies together to form *modules*.

A module is defined as a connected volume, limited only by quadric surfaces, that contains one or several bodies. A module can contain other modules, which will be referred to as *submodules* of the first. The volume of a module is filled with a homogeneous material, which automatically fills the cavities of the module (*i.e.*, volumes that do not correspond to a body or to a submodule); these filled cavities are considered as a single new body. A body that is connected and limited only by surfaces can be declared either as a body or as a module. For the sake of simplicity, modules are required to satisfy the following conditions: 1) the bodies and submodules of a module must be completely contained within the parent module (*i.e.*, portions of bodies or submodules that lie outside the module are not allowed) and 2) a submodule of a module cannot overlap with other submodules and bodies of the same module (this is necessary to make sure that a particle can only enter or leave a module through its limiting surfaces). The bodies of a module are still assumed to be defined in ascending order, *i.e.*, a body is limited by its surfaces and by the previously defined bodies *of the same module*, so that inclusions and interpenetrating bodies can be easily defined.

A module (with its possible submodules) can represent a rigid part (*e.g.*, a radioactive source, an accelerator head, a detector, a phantom, etc.) of a more complex material system. To facilitate the definition of the geometry, PENGEOM allows free translations and rotations of individual modules. Thus, the definition of a module (see Table 2 below) includes the parameters of a rotation (defined by its Euler angles) and a translation, which are optional and serve to modify the position and orientation of the module (and all its bodies and submodules) with respect to the laboratory reference frame. As in the case of surfaces in implicit form, the rotation is applied first. It is worth noticing that the translation or rotation of a module affects its defining surfaces and bodies; the original definitions of these elements are lost. If these surfaces and bodies are further used to define other bodies or modules they will be employed in their transformed forms.

In practical simulations with finite geometries, the tracking of a particle should be discontinued when it leaves the material system. In PENGEOM this is done automatically by assuming that the complete system is contained within a single convex module, the *enclosure*, which comprises the whole system. It is also convenient (but not necessary) to require that the enclosure has a finite volume, so that all rays starting from any point within the volume of the enclosure do intersect one of its limiting surfaces at a finite distance. When the whole geometry is contained inside a single module, PENGEOM identifies this module as the enclosure and assumes that it is convex. When an enclosure is not defined by the user (*i.e.*, when the geometry consists of several separate modules and/or bodies that are not inside a single module), PENGEOM defines the enclosure as a large sphere of radius 10^7 length units, centered at the origin of coordinates. It is assumed that there is a perfect vacuum outside the enclosure, and in any inner volume that is not a body or a filled module. If the geometry definition contains bodies that extend beyond the enclosure, they are truncated and only the parts inside the enclosure are retained. Hence, particles that leave the enclosure will never return to the material system.

For programming purposes, it is useful to consider each module as the mother of its bodies and submodules, and as the daughter of the module that contains it. We thus have a genealogical tree with various generations of modules and bodies. The first generation reduces to the enclosure, which is the only motherless module. The n-th generation consists of modules and bodies whose mothers belong to the $(n-1)$-th generation. The structure of each module is defined by its limiting surfaces (which determine the border with the external world) and those of their descendants (which determine the module's internal structure).

The benefit of using modules is that, while a particle is moving within a module, the only accessible bodies are the daughters (bodies and submodules) of that module. This effectively limits the number of surfaces that need to be analyzed at each move. Only when the particle crosses one of the limiting surfaces of the current module, do we need to consider the outer geometry. In addition, when a particle leaves a module, it remains within the volume of either the parent module or one of its ancestors. The numerical work needed to locate and track particles may thus be largely reduced by defining a well-ramified genealogical tree of modules. Optimal organization is obtained when each module has a relatively simple inner structure with a small number of daughters. Practical experience indicates that tracking particles through complex quadric geometries may be faster than with elaborate ray-tracing methods, such as octree encoding [33; 34], provided that the tree of modules is finely ramified.

4. Geometry definition file

The geometry is defined from a formatted input text file, which consists of a series of blocks defining the different elements (surfaces, bodies and modules). A definition block consists of a number of strictly formatted text lines; it starts and ends with a separation line filled with zeros (see Table 2). The first line in a block starts with one of the defining 8-character strings "`SURFACE␣`", "`SURFACE*`", "`BODY␣␣␣␣`", "`MODULE␣␣`", `CLONE␣␣␣`", "`INCLUDE␣`", "`INCLUDE*`" or "`END␣␣␣␣␣`". A line with the string "`END␣␣␣␣␣`" after a separation line discontinues the reading of geometry data. Each element is identified by its type (surface, body or module) and a four-character string, the user label, which designates the elements within the definition file. In the PENGEOM subroutines, geometry elements are identified with a numerical label: elements of a given type are numbered consecutively, according to their input order. Bodies and modules are considered as elements of the same type (*i.e.*, assigning the same user label to a body and to a module will cause an error of the reading routine).

In the input file, numerical quantities must be written within the parentheses in the specified format. Lengths are in arbitrary units; angles can be given in either degrees (`DEG`) or radians (`RAD`). When angles are in degrees, it is not necessary to specify the unit. Numerical parameters are assigned the default values indicated in Table 2; lines that define parameters with their default values can be removed from the definition file. Each numerical parameter is followed by an `I4` value, which must be set equal to zero or negative to make the parameter value effective. When this field contains a positive integer, the parameter value can be modified through the initialization subroutine `GEOMIN`. This permits the user to modify the geometry from the main program (*e.g.*, translate or rotate a module).

Surfaces can be defined either in reduced or implicit form. Surface parameters are optional and can be entered in any order. The keyword `SURFACE*` serves to define "fixed" surfaces, which will not be affected by possible translations or rotations in subsequent stages of the geometry definition. These surfaces are useful, *e.g.*, to define aligned modules.

Bodies can be delimited by previously defined surfaces, bodies and modules. The material number (2nd line in the block, an integer) must agree with the convention adopted in the simulation. Void inner volumes can be described as material bodies with `MATERIAL` set equal to 0 (or a negative number). A line is required to define each limiting surface, with its side pointer, and each limiting body or module.

The definition of a module (see Table 2) consists of its limiting surfaces and side pointers, and the set of inner bodies and submodules. The inner cavities (if any) are filled with the declared material. Rotation and translation are optional and apply to all elements of the module, except the starred surfaces. The line filled with `1`'s ends the definition of elements and starts that of transformation parameters; it can be omitted if no transformation parameters are entered. The `CLONE` operation automatically clones a module (with its submodules and inner bodies) and changes the position and orientation of the cloned module. This operation is helpful to define systems with repeated structures, such as array detectors and multi-leaf collimators. Notice that the cloned module cannot overlap any of the previously defined bodies.

The `INCLUDE` option allows a predefined structure (*e.g.*, a scintillation

Table 2: Formats of the geometry definition blocks and default parameter values.

```
0000000000000000000000000000000000000000000000000000000000000000
SURFACE (  A4)  reduced form
INDICES=(I2,I2,I2,I2,I2)
X-SCALE=(        E22.15        ,  I4)              (DEFAULT=1.0)
Y-SCALE=(        E22.15        ,  I4)              (DEFAULT=1.0)
Z-SCALE=(        E22.15        ,  I4)              (DEFAULT=1.0)
  OMEGA=(        E22.15        ,  I4) DEG          (DEFAULT=0.0)
  THETA=(        E22.15        ,  I4) DEG          (DEFAULT=0.0)
    PHI=(        E22.15        ,  I4) RAD          (DEFAULT=0.0)
X-SHIFT=(        E22.15        ,  I4)              (DEFAULT=0.0)
Y-SHIFT=(        E22.15        ,  I4)              (DEFAULT=0.0)
Z-SHIFT=(        E22.15        ,  I4)              (DEFAULT=0.0)
0000000000000000000000000000000000000000000000000000000000000000
SURFACE (  A4)  implicit form
INDICES=( 0, 0, 0, 0, 0)
    AXX=(        E22.15        ,  I4)              (DEFAULT=0.0)
    AXY=(        E22.15        ,  I4)              (DEFAULT=0.0)
    AXZ=(        E22.15        ,  I4)              (DEFAULT=0.0)
    AYY=(        E22.15        ,  I4)              (DEFAULT=0.0)
    AYZ=(        E22.15        ,  I4)              (DEFAULT=0.0)
    AZZ=(        E22.15        ,  I4)              (DEFAULT=0.0)
     AX=(        E22.15        ,  I4)              (DEFAULT=0.0)
     AY=(        E22.15        ,  I4)              (DEFAULT=0.0)
     AZ=(        E22.15        ,  I4)              (DEFAULT=0.0)
     A0=(        E22.15        ,  I4)              (DEFAULT=0.0)
1111111111111111111111111111111111111111111111111111111111111111
  OMEGA=(        E22.15        ,  I4) DEG          (DEFAULT=0.0)
  THETA=(        E22.15        ,  I4) DEG          (DEFAULT=0.0)
    PHI=(        E22.15        ,  I4) RAD          (DEFAULT=0.0)
X-SHIFT=(        E22.15        ,  I4)              (DEFAULT=0.0)
Y-SHIFT=(        E22.15        ,  I4)              (DEFAULT=0.0)
Z-SHIFT=(        E22.15        ,  I4)              (DEFAULT=0.0)
0000000000000000000000000000000000000000000000000000000000000000
BODY    (  A4)  Text describing the body ...
MATERIAL(  A4)
SURFACE (  A4), SIDE POINTER=( 1)
BODY    (  A4)
MODULE  (  A4)
0000000000000000000000000000000000000000000000000000000000000000
MODULE  (  A4)  Text describing the module ...
MATERIAL(  A4)
SURFACE (  A4), SIDE POINTER=( 1)
BODY    (  A4)
MODULE  (  A4)
1111111111111111111111111111111111111111111111111111111111111111
  OMEGA=(        E22.15        ,  I4) DEG          (DEFAULT=0.0)
  ...
Z-SHIFT=(        E22.15        ,  I4)              (DEFAULT=0.0)
0000000000000000000000000000000000000000000000000000000000000000
CLONE   (  A4)  Copies a module and moves it
MODULE  (  A4)  Original module
1111111111111111111111111111111111111111111111111111111111111111
  OMEGA=(        E22.15        ,  I4) DEG          (DEFAULT=0.0)
  ...
Z-SHIFT=(        E22.15        ,  I4)              (DEFAULT=0.0)
0000000000000000000000000000000000000000000000000000000000000000
INCLUDE
   FILE=(filename.ext)
0000000000000000000000000000000000000000000000000000000000000000
END      0000000000000000000000000000000000000000000000000000000
```

detector, an encapsulated radioactive source, . . .) to be inserted within the geometry file. The inserted structure is defined by a complete definition file (*i.e.*, ending with an "`END␣␣␣␣␣`" line). The inserted structure cannot overlap any of the existing bodies.

PENGEOM admits up to 10,000 surfaces and 5,000 bodies and modules. A single surface can be used to define several bodies and/or modules. Conversely, the same surface can be defined several times, *e.g.*, to keep the definition of a body or module complete and independent of those of other bodies and modules. The unnecessary duplication of a surface does not affect the simulation speed, because PENGEOM identifies "redundant" surfaces and effectively removes them.

5. The subroutine package PENGEOM

The Fortran subroutine package PENGEOM consists of several subroutines, which perform geometrical operations guided by a steering main program. Most of the input/output of these subroutines is through the Fortran module `TRACK_mod`, which contains the coordinates of the particle, $\mathbf{r} = (\mathtt{X}, \mathtt{Y}, \mathtt{Z})$, the direction cosines of the direction of movement, $\hat{\mathbf{d}} = (\mathtt{U}, \mathtt{V}, \mathtt{W})$, the label of the current body, `IBODY`, and the material `MAT` (the code number employed in the physics simulation program) in that body.

The subroutines to be invoked from the main program are the following;

• `SUBROUTINE GEOMIN`

Reads the geometry definition file and initialises the geometry package. It labels the various kinds of elements (surfaces, bodies, and modules) in strictly increasing order; it also allows redefining some of the geometry parameters. During simulation, geometry elements are identified by the labels assigned by PENGEOM, which may be different from the user labels in the geometry definition file.

• `SUBROUTINE LOCATE`

Determines the body `IBODY` that contains the point with coordinates `(X,Y,Z)`, and the material `MAT` in that body. The output value `MAT` $= 0$ indicates that the particle is in a void region.

• `SUBROUTINE STEP(DS,DSEF,NCROSS)`

This subroutine performs the geometrical part of the track simulation under the guidance of the steering main program, which determines the direction of motion `(U,V,W)` and the length `DS` of each free flight. `STEP` moves that particle from its initial position `(X,Y,Z)` in the direction `(U,V,W)` and stops it at the end of the step, or just after entering a new material (particles are not halted at “interfaces” between bodies of the same material). The output values `DSEF` and `NCROSS` are, respectively, the distance travelled within the initial material and the number of interface crossings (`NCROSS=0` if the particle does not leave the initial material, greater than `0` if the particle enters a new material or leaves the enclosure). If the particle enters a void region, `STEP` continues the particle track, as a straight segment, until it penetrates a material body or leaves the system (the path length through inner void regions is not included in `DSEF`). When the particle arrives from a void region (`MAT` $=$ `0`), it is stopped just after entering the first material body. At output, `(X,Y,Z)` are the coordinates of the final position and `IBODY` and `MAT` are the corresponding body and material. The output value `MAT` $=$ `0` indicates that the particle has escaped from the system.

In its normal operation mode, `STEP` does not stop particles at surfaces that limit adjacent bodies of the same material, because this would slow down the simulation unnecessarily. Therefore, these surfaces are “invisible” from the main program. To extract information about particle fluxes within the geometrical structure, the user can define *impact detectors*. Each impact detector consists of a set of material bodies, which are assigned the same detector label. When a transported particle enters a body from vacuum or from another body that is not part of the detector, subroutine `STEP` halts the particle at the surface of the active body and control is returned to the main program. Thus, if two adjacent bodies of the same material pertain to different detectors, their common limiting surface becomes visible. In practical simulations of radiation transport, impact detectors can be used to tally the energy spectrum and the angular distribution of “detected” particles (*i.e.*, particles that enter an active body) or, more specifically, to generate a phase-space file, where the state variables of particles at the detector entrance are recorded.

6. The Java graphical user interface PenGeomJar

The reliability of simulations with complex material systems rests heavily on the correct specification of the geometry, which can only be verified by direct visual inspection. Elaborate graphics tools capable of rendering quadric geometries are freely available [*e.g.*, POVRay (`http://www.povray.org/`), OpenGL (`http://www.opengl.org/`), ...], although most of them implement a logical structure based on surfaces rather than solid volumes. Because

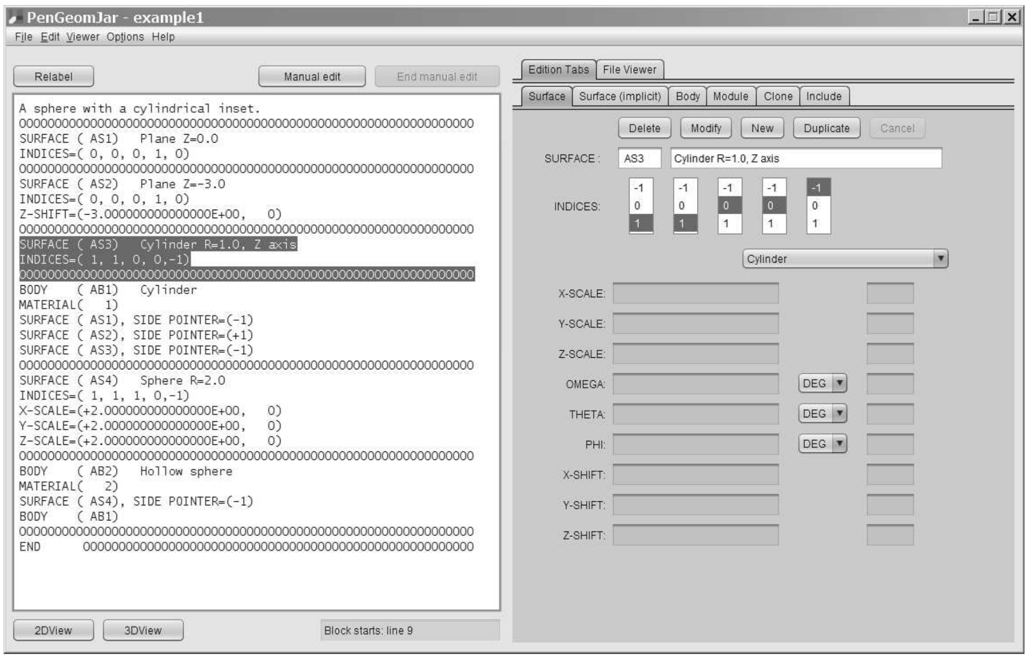
PENGEOM uses bodies as basic geometry elements, and the definition sequence and organization is tailored to optimize simulation speed, the translation of the geometry to the format of other programs is not easy and, moreover, it implies a risk of introducing distortions (such as overlaps of different material bodies and incorrect material assignments).

To ensure a faithful graphical representation, and to help the user to debug the definition files, we have developed specific software which performs the geometry rendering by using the PENGEOM subroutines. Thus, the images shown on the screen correspond to the geometry actually passed to the simulation program. The distribution packages of the various versions of the PENELOPE code include a pair of executable binaries named GVIEW2D and GVIEW3D that generate two- and three-dimensional 24-bit colour images of PENGEOM geometries. These binaries run on personal computers under Microsoft Windows, or on other platforms with a Windows emulator.

Aiming at a wider portability, we have written a Java program, named PenGeomJar, which provides a graphical user interface (GUI) from where we can edit the geometry definition files and run the two- and three-dimensional viewers. The main window of the GUI contains a menu bar, a text panel, a block-definition area, and rendering buttons (see Fig. 2). The right panel of the window is the definition area, with tabs for the various definition blocks (surface, body, module, clone, and include). The left panel is the editing area, it displays the actual contents of the geometry definition file, where it can be edited automatically or manually.

The GUI allows the user to generate two- and three-dimensional images of the geometry during edition of the definition file. Rendering stops either when an input format is incorrect (reading error) or when a clear inconsistency in the definition file is found (*e.g.*, when the element that is being defined and the furnished information do not match). The GUI displays error messages and, in addition, indicates the line of the geometry definition file where the error has been found. Typical syntax errors can be readily corrected by editing the geometry file. Once the structure of the input file is correct, the program does not stop and the geometry is displayed for further analysis.

Figure 3 shows 2D and 3D images of an example quadric geometry included in the distribution package, a mathematical anthropomorphic phantom representing an adult, in which organs with the average shapes and dimensions are defined by quadric surfaces [35; 36; 37; 38]. This kind of phantom is used in Monte Carlo dosimetry studies [39] because it allows fast calculation of



```
A sphere with a cylindrical inset.
0000000000000000000000000000000000000000000000000000000000000000
SURFACE ( AS1)   Plane Z=0.0
INDICES=( 0, 0, 0, 1, 0)
0000000000000000000000000000000000000000000000000000000000000000
SURFACE ( AS2)   Plane Z=-3.0
INDICES=( 0, 0, 0, 1, 0)
Z-SHIFT=(-3.000000000000000E+00,   0)
0000000000000000000000000000000000000000000000000000000000000000
SURFACE ( AS3)   Cylinder R=1.0, Z axis
INDICES=( 1, 1, 0, 0,-1)
0000000000000000000000000000000000000000000000000000000000000000
BODY    ( AB1)   Cylinder
MATERIAL(   1)
SURFACE ( AS1), SIDE POINTER=(-1)
SURFACE ( AS2), SIDE POINTER=(+1)
SURFACE ( AS3), SIDE POINTER=(-1)
0000000000000000000000000000000000000000000000000000000000000000
SURFACE ( AS4)   Sphere R=2.0
INDICES=( 1, 1, 1, 0,-1)
X-SCALE=(+2.000000000000000E+00,   0)
Y-SCALE=(+2.000000000000000E+00,   0)
Z-SCALE=(+2.000000000000000E+00,   0)
0000000000000000000000000000000000000000000000000000000000000000
BODY    ( AB2)   Hollow sphere
MATERIAL(   2)
SURFACE ( AS4), SIDE POINTER=(-1)
BODY    ( AB1)
0000000000000000000000000000000000000000000000000000000000000000
END      0000000000000000000000000000000000000000000000000000000
```

Block starts: line 9

SURFACE : AS3 Cylinder R=1.0, Z axis

INDICES: 1, 1, 0, 0, -1

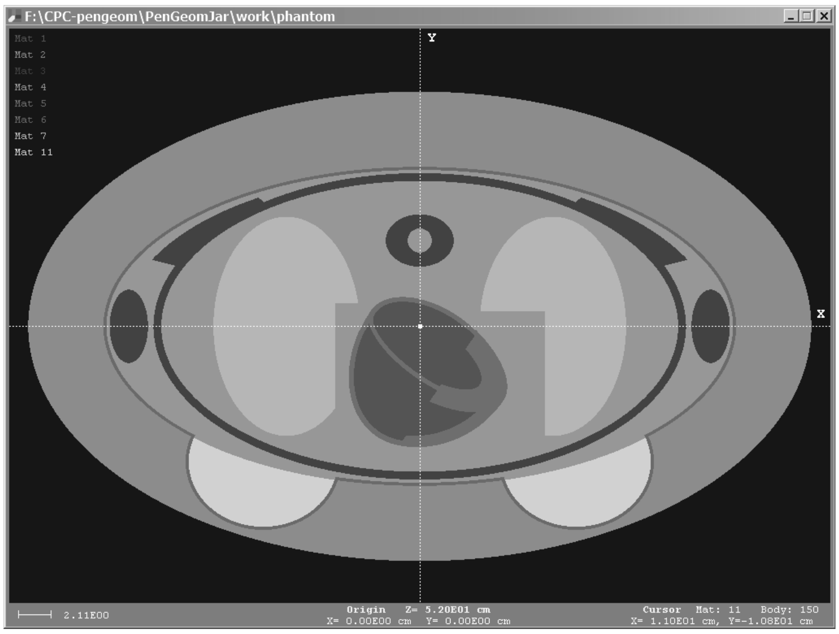
Cylinder

Figure 2: Main window of the GUI PenGeomJar.

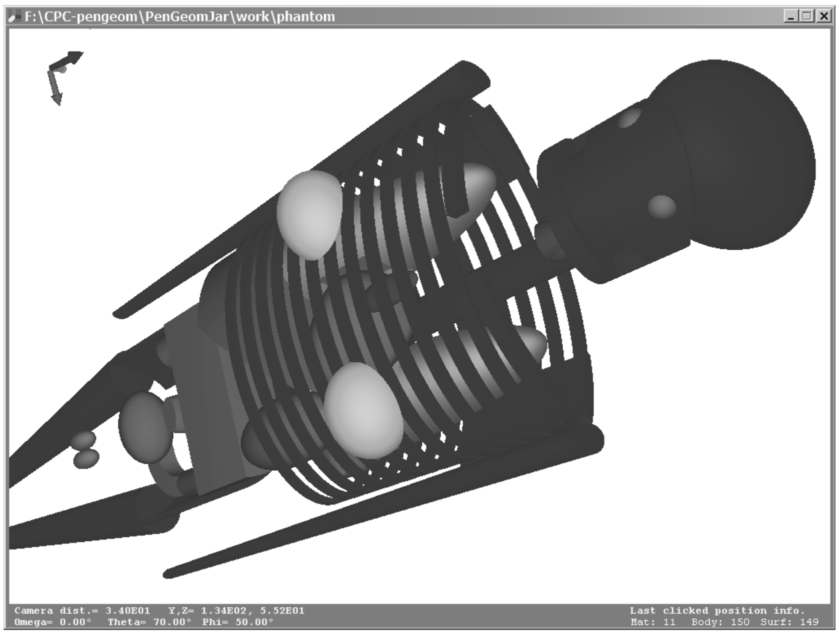
ray-surface intersections[1]

The method used to generate a two-dimensional image (2DView) consists of following a particle that moves on a plane perpendicular to an axis of the reference frame, which is mapped on the window. The particle starts from a position that corresponds to the leftmost pixel of each row and moves along a straight trajectory to the right of the window. To do this, subroutine `STEP` is called repeatedly, maintaining the direction of movement and with a large value of the step length `DS` (so that each body is crossed in a single step). A colour code is assigned to each material or body, and pixels are lit up with the active colour when they are crossed by the particle trajectory. The final picture is a map of the bodies or materials intersected by the window plane (Fig. 3). The orientation of the window plane, as well as the position and size of the window view, may be changed interactively. The body labels shown on

[1]More elaborate computational phantoms of the human body utilize boundary representation methods, with organs limited by smooth surfaces described either by means of non-uniform rational B-splines or polygon mesh surfaces (see, e.g., [40]).



```
2.11E00          Origin    Z= 5.20E01 cm          Cursor  Mat: 11   Body: 150
          X= 0.00E00 cm  Y= 0.00E00 cm       X= 1.10E01 cm, Y=-1.08E01 cm
```



```
Camera dist.= 3.40E01   Y,Z= 1.34E02, 5.52E01        Last clicked position info.
Omega= 0.00°  Theta= 70.00°  Phi= 50.00°             Mat: 11  Body: 150  Surf: 149
```

Figure 3: Top: 2DView transverse section of the anthropomorphic phantom, perpendicular to the z axis at the level of the heart and breast. Bottom: 3DView of the same phantom with the materials air, skin and soft tissue made transparent, *i.e.*, showing only the skeleton and the inner organs.

the screen are the ones assigned by PENGEOM. These labels are needed for defining impact detectors, and for scoring purposes (*e.g.*, to determine the distribution of energy deposited within a particular body). A text block in the graphics window shows the material and body at the cursor position.

The renderer 3DVIEW generates three-dimensional pictures of the geometry by using a simple ray-tracing algorithm, with the source light and the camera at the same position. Images can be modified by using the context menu or by entering one-character commands directly from the graphics window. Bodies and materials are displayed with the same colour code as in 2DVIEW and the intensity of each pixel is determined by the angle between the vision line (which starts at the camera position and goes through the pixel centre) and the normal to the limiting surface that is intersected first. This method does not produce shadows and disregards light diffusion, but makes fairly realistic three-dimensional images and is quite fast. To reveal the inner structure of the system, the program can eliminate a wedge (limited by two vertical planes that intersect in the z-axis); alternatively, the user can select specific materials and make them transparent. The different surfaces and bodies shown in the image can be identified by clicking with the mouse (their labels are displayed in the lower left corner of the graphics window when the information bar is active). It is worth noting that 3DVIEW generates the image pixel by pixel, whereas 2DVIEW does it by drawing straight lines on the render window; as a result, 2DVIEW is much faster.

It should be mentioned that the use of fuzzy quadric surfaces in the current version of PENGEOM improves the resolution over previous versions of the subroutines which used static surfaces. As a protection against round-off errors, at each surface crossing the old subroutines shifted the particle about 10^{-8} length units beyond the surface; this extra shift blurred all surfaces and limited the resolution. The viewers 2DVIEW and 3DVIEW now correctly render very small objects (*e.g.*, a spherical shell with inner radius of 10^{-9} units and a thickness of 5×10^{-11} units) as well as large structures with small details (*e.g.*, a spherical shell of 10^6 radius and 10^{-5} thickness), which were not correctly resolved by older versions of PENGEOM.

7. Distribution package

The code system is distributed in two separate zip-compressed files. The first of these files, `pengeom.zip`, has a root folder named `pengeom` that contains the pdf file of the manual (`pengeom-manual.pdf`) and the following

subfolders:
1) `source`: contains the source file `pengeom.f` and a layout geometry file.
2) `examples`: examples of geometry definition files.
3) `pen2pov`: contains tools for translating PENGEOM definition files to POV-RAY format, and for rendering realistic three-dimensional geometry images with POV-RAY. More information on these tools is given in the manual.
4) `src`: This directory contains a complete set of source files to generate the libraries used by the executable `pengeom.jar` for any UNIX-like platform having the corresponding compilers and Java Development Kit installed. Instructions for compiling the Fortran subroutines and the linking C code are provided in the file `readme.txt`.

Because PENGEOMJAR links Java methods to the PENGEOM Fortran subroutines, different executable binaries are provided for Windows (32 bits and 64 bits) and Linux operating systems. The distribution package contains a set of zip-compressed files, one for each supported operating system, named `PenGeomJar_OSver.zip` or `PenGeomJar_OSver.tar.gz`. Each of these files contains a single folder named `PenGeomJar` with the Java program `pengeom.jar` generated for the corresponding operating system, a subfolder, and a number of auxiliary files. The program is stand-alone (it does not need installation) and only requires the Java Runtime Environment (JRE) to be installed on the computer. Once the contents of the appropriate distribution file is copied in the computer disk, the program is started either by running a simple batch file (`run.bat` and `run.sh` in Windows and Linux, respectively), by issuing the command "`java -jar pengeom.jar`", or by double-clicking on the `pengeom.jar` icon if the extension `.jar` is associated to Java.

Acknowledgements

We are indebted to Josep Sempau for his work in the early development of PENGEOM and for many valuable suggestions that have led to improvements of the tracking algorithm. Financial support from the Spanish Ministerio de Economía y Competitividad and ERDF (project no. FPA2013-44549-P) and from the Generalitat de Catalunya (grant 2014 SGR 846) is gratefully acknowledged. F. S.-P. appreciates support from the Alexander von Humboldt Foundation through a Humboldt Research Fellowship.

Refrences

[1] F. Salvat, J. M. Fernández-Varea, J. Sempau, PENELOPE-2011: A code System for Monte Carlo Simulation of Electron and Photon Transport, OECD/NEA Data Bank, Issy-les-Moulineaux, France, 2011, available from `http://www.nea.fr/lists/penelope.html`.

[2] M. J. Berger, Monte Carlo calculation of the penetration and diffusion of fast charged particles, in: B. Alder, S. Fernbach, M. Rotenberg (Eds.), *Methods in Computational Physics*, Vol. 1, Academic Press, New York, 1963, pp. 135–215.

[3] M. J. Berger, S. M. Seltzer, An overview of ETRAN Monte Cario methods, in: T. M. Jenkins, W. R. Nelson, A. Rindi (Eds.), *Monte Carlo Transport of Electrons and Photons*, Plenum, New York, 1988, Ch. 7.

[4] M. J. Berger, S. M. Seltzer, Applications of ETRAN Monte Cario codes, in: T. M. Jenkins, W. R. Nelson, A. Rindi (Eds.), *Monte Carlo Transport of Electrons and Photons*, Plenum, New York, 1988, Ch. 9.

[5] M. J. Berger, S. M. Seltzer, ETRAN — Experimental Benchmarks, in: T. M. Jenkins, W. R. Nelson, A. Rindi (Eds.), *Monte Carlo Transport of Electrons and Photons*, Plenum, New York, 1988, Ch. 8.

[6] J. A. Halbleib, R. P. Kensek, T. A. Mehlhorn, G. D. Valdez, S. M. Seltzer, M. J. Berger, ITS version 3.0: the integrated TIGER series of coupled electron/photon Monte Carlo transport codes, Tech. Rep. SAND91-1634, Sandia National Laboratories, Albuquerque, NM (1992).

[7] W. R. Nelson, H. Hirayama, D. W. O. Rogers, The EGS4 Code System, Tech. Rep. SLAC-265, Stanford Linear Accelerator Center, Stanford, California (1985).

[8] R. Brun, F. Bruyant, M. Maire, A. C. McPherson, P. Zanarini, GEANT3, Tech. Rep. DD/EE/84–1, CERN, Geneva (1989).

[9] I. Kawrakow, D. W. O. Rogers, The EGSnrc code system: Monte Carlo simulation of electron and photon transport, Tech. Rep. PIRS-701, National Research Council of Canada, Ottawa (2001).

[10] X-5 Monte Carlo Team, MCNP—A general Monte Carlo N-particle transport code, version 5, Report LA-UR-03-1987, Los Alamos National Laboratory, Los Alamos, NM, 2003.

[11] S. Agostinelli, J. Allison, K. Amako, J. Apostolakis, H. Araujo, P. Arce, M. Asai, Geant4—a simulation toolkit, Nucl. Instrum. Meth. A 506 (2003) 250–303.

[12] J. Allison, K. Amako, J. Apostolakis, H. Araujo, P. Arce Dubois, M. Asai, Geant4 developments and applications, IEEE Trans. Nucl. Sci. 53 (2006) 270–278.

[13] A. Ferrari, P. R. Sala, A. Fassò, J. Ranft, Fluka: a multi-particle transport code, Tech. Rep. CERN200500X, INFN TC 05/11, SLACR773, CERN, Geneva (2005).

[14] H. Hirayama, Y. Namito, A. F. Bielajew, S. J. Wilderman, W. R. Nelson, The EGS5 Code System, Tech. Rep. SLAC-R-730 (KEK 2005-8), Stanford Linear Accelerator Center, Menlo Park, California (2006).

[15] J. T. Goorley, M. R. James, T. E. Booth, F. B. Brown, J. S. Bull, L. J. Cox, J. W. D. Jr., J. S. Elson, Initial MCNP6 Release Overview - MCNP6 version 1.0, LA-UR-13-22934, Los Alamos National Laboratory, Los Alamos, NM, 2013.

[16] T. M. Jenkins, W. R. Nelson, A. Rindi, Monte Carlo Transport of Electrons and Photons, Plenum, New York, 1988.

[17] J. Baró, J. Sempau, J. M. Fernández-Varea, F. Salvat, PENELOPE: An algorithm for Monte Carlo simulation of the penetration and energy loss of electrons and positrons in matter, Nucl. Instrum. Meth. B 100 (1995) 31–46.

[18] F. Salvat, A generic algorithm for Monte Carlo simulation of proton transport, Nucl. Instrum. Meth. B 316 (2013) 144–159.

[19] J. M. Fernández-Varea, R. Mayol, J. Baró, F. Salvat, On the theory and simulation of multiple elastic scattering of electrons, Nucl. Instrum. Meth. B 73 (1993) 447–473.

[20] I. Kawrakow, A. F. Bielajew, On the condensed history technique for electron transport, Nucl. Instrum. Meth. B 142 (1998) 253–280.

[21] A. F. Bielajew, F. Salvat, Improved elctron transport mechanics in the PENELOPE Monte Carlo model, Nucl. Instrum. Meth. B 173 (2001) 332–343.

[22] H. Nikjoo, S. Uehara, I. G. Khvostunov, F. A. Cucinotta, W. E. Wilson, D. T. Goodhead, Monte Carlo track structure for radiation biology and space applications, Physica Medica XVII, Supplement 1 (2001) 38–43.

[23] R. Gauvin, E. Lifshin, H. Demers, P. Horny, H. Campbell, Win X-ray: A new Monte Carlo program that computes x-ray spectra obtained with a scanning electron microscope, Microsc. Microanal. 12 (2006) 49–64.

[24] N. W. M. Ritchie, A new Monte Carlo application for complex sample geometries, Surf. Interface Anal. 37 (2005) 1006–1011.

[25] R. Shimizu, Z.-J. Ding, Monte Carlo modelling of electron-solid interactions, Rep. Prog. Phys. 55 (1992) 487–531.

[26] M. Dapor, Monte Carlo simulation of backscattered electrons and energy from thick targets and surface films, Phys. Rev. B 46 (1992) 618–625.

[27] W. S. M. Werner, Electron transport in solids for quantitative surface analysis, Surf. Interface Anal. 31 (2001) 141–176.

[28] A. Jablonski, C. J. Powell, S. Tanuma, Monte Carlo strategies for simulations of electron backscattering from surfaces, Surf. Interface Anal. 37 (2005) 861–874.

[29] K. O. Jensen, A. B. Walker, Monte Carlo simulation of the transport of fast electrons and positrons in solids, Surf. Sci. 292 (1993) 83–97.

[30] J. M. Fernández-Varea, D. Liljequist, S. Csillag, R. Räty, F. Salvat, Monte Carlo simulation of 0.1–100 keV electron and positron transport in solids using optical data and partial wave methods, Nucl. Instrum. Meth. B 108 (1996) 35–50.

[31] A. F. Bielajew, `HOWFAR` and `HOWNEAR`: Geometry modeling for Monte Carlo particle transport, Tech. Rep. PIRS-0341, National Research Council of Canada, Ottawa (1995).

[32] A. R. Edmonds, Angular Momentum in Quantum Mechanics, Princeton University Press, Princeton, NJ, 1960.

[33] D. Meagher, Geometric modeling using octree encoding, Computer Graphics and Image Processing 19 (1982) 129–147.

[34] A. S. Glassner, Space subdivision for fast ray tracing, IEEE Computer Graphics and Applications 4 (1984) 15–22.

[35] M. Cristy, K. F. Eckerman, Specific absorbed fractions of energy at various ages from internal photon sources I. Methods, Tech. Rep. ORNL/TM 8381/Vi, Oak Ridge National Laboratory, Oak Ridge, TN (1987).

[36] A. V. Ulanovsky, K. F. Eckerman, Absorbed fractions for electron and photon emissions in the developing thyroid: fetus to five years old, Radiation Protection Dosimetry 79 (1998) 419–424.

[37] A. V. Ulanovsky, K. F. Eckerman, Modifications to the ORNL phantom series in simulation of the responses of thyroid detectors, Radiation Protection Dosimetry 79 (1998) 429–431.

[38] L. G. Bouchet, W. E. Bolch, D. A. Weber, H. L. Atkins, J. W. B. sr, MIRD Pamphlet No. 15: radionuclide S values in a revised dosimetric model of the adult head and brain, The Journal of Nuclear Medicine 40 (1999) 62S–101S.

[39] G. Díaz-Londoño, S. García-Pareja, F. Salvat, A. M. Lallena, Monte Carlo calculation of specific absorbed fractions: variance reduction techniques, Phys. Med. Biol. 60 (2015) 2625–2644.

[40] Y. Na, B. Zhang, J. Zhang, P. F. Caracappa, X. G. Xu, Deformable adult human phantoms for radiation protection dosimetry: anthropometric data representing size distributions of adult worker populations and software algorithms, Phys. Med. Biol. 55 (2010) 3789–3811.