A geometry modelling system for ray tracing or particle transport Monte Carlo simulation

Lucreţiu M. Popescu

National Institute of R & D for Environment, Environmental Radioactivity Laboratory, 294 Spl. Independenţei, 77703 Bucharest, Romania [1]

Abstract

The principles of a geometry modelling system for ray tracing or Monte Carlo particle transport simulation are presented. The model uses the boundary representation of volumes and was developed using object oriented programming. The surface is adopted as the basic element in the model. Complex structures are described using the derived concept of compound surfaces. Particle tracking exceptions caused by floating point rounding errors are discussed and robust algorithms that use geometrical reasoning to address these problems are presented.

Key words: Monte Carlo, particle transport, geometry modelling, boundary representation, robust algorithms.
PACS: 02.50.N, 29.27.Eg, 41.85.Ja, 87.53.Wz

1 Introduction

One of the advantages of using the Monte Carlo method to solve particle transport problems is the fact that it does not rely on any theoretical limitations imposed by geometry or material composition, offering the possibility to obtain solutions

[1] Present address: University of Pennsylvania, Department of Radiology, 423 Guardian Drive, 4th Floor Blockley Hall, Philadelphia, PA 19104-6021, USA; e-mail: popescu@mipg.upenn.edu

for various combinations of materials and volume shapes. However, to take full advantage of this, a Monte Carlo simulation system should offer the user a way of representing various possible shapes and combinations of geometries in a flexible, easy to use and efficient manner. For this reason a very important role in a Monte Carlo particle transport application is played by the geometry modelling system.

In order to allow the selection of the proper set of data necessary for transport simulation (i.e. cross sections, specific transport algorithms) and to determine certain actions, like tallying the values of the quantities of interest, particle positions must be described not only physically using their spatial coordinates but also from a topological point of view in relation with the surrounding geometrical entities.

According to the usual particle transport Monte Carlo simulation schemes a geometry representation system must provide the following basic functions.

- To verify if a point is inside a certain geometrical domain (volume).
- The particle being inside a certain volume, to determine the distance to the boundary surface along the particle trajectory, or otherwise said, to determine the exit point without interacting in the medium.
- When the particle leaves the current volume to determine the new volume into which it enters, or to flag that the particle has left the domain of interest.
- To determine the distance to the volume boundary (as the distance from the current position to the closest point on the boundary). This is often necessary for tallying purposes or for optimization reasons in the case when lateral displacement corrections must be applied for long steps as in the case of electron condensed history schemes.

A first way to deal with these problems in simulation software consists of defining only the prototypes (the interfaces) of the required functions while their specific implementation remains the responsibility of the user, as in the case of the basic distribution of the EGS system [1]. Additionally the user can be helped by providing a library with predefined geometrical objects [2,3]. This approach has the advantage that it can lead to an optimized and fast code. However this advantage can fade out if we take into account the effort and the time required for writing and testing the new routines for each new problem.

In a comprehensive approach, the geometry representation system and the algorithms should be devised in such a way as to reduce the user's intervention to a

minimum, namely the description of the geometry as input data or as an initialization routine. In such a system all geometrical problems should be handled automatically dependent only on the particle position, moving direction and the initial description of the geometry.

The difficulty with this approach consists of the fact that a number of constraints must be satisfied.

- First a number of ambiguities that may occur in the geometrical description of the problem must be resolved.
- The system must be robust. The topological description of the particle position must not be affected by floating point rounding errors or exceptions that may occur during the calculation of the distances and coordinates.
- The algorithms should be fast. The computing overhead necessary to assure the generality of the modelling system should be kept in reasonable limits.

Comprehensive geometry modelling systems were introduced in several particle transport Monte Carlo simulation codes, like MCNP [4] or GEANT [5], well known in the field of nuclear physics or high energy physics, or medical physics. Reviews on this subject are presented in [6,7].

One can distinguish between two main approaches, the Constructive Solid Geometry (CSG) modelling or Combinatorial Geometry (CG) and surface or boundary representation (B-rep) modelling. In the CSG systems complex geometrical bodies are described using Boolean algebra with a limited set of predefined bodies as in GEANT3 [5] or in [8]. In the systems that use boundary representation the geometrical objects are described using Boolean algebra with space regions delimited by predefined surfaces as in the case of MCNP [4] or version 4 of GEANT that partially evolved toward boundary representation [9].

The requirements listed above are satisfied at different levels by the different systems for geometry modelling. With the CSG systems it is usually more easy to build well defined geometrical structures, whereas the boundary representation systems are more prone to lead to ambiguous, incompletely defined, or degenerate structures. The complex systems designed to cover with higher generality the multitude of cases that may occur have a higher computing overhead and thus in many applications perform significantly slower than systems implementing simpler solutions.

It is often the case that there is little information available about the robustness of a particular geometry modelling system. Sometimes this issue is little or not at all documented and the user is the one who has to deal with it. In other cases the authors acknowledge the problem and offer ad-hoc solutions to overcome it. The robustness issue in geometry based programs has received increased attention in recent years [10–14]. So far, however, there has been little systematic effort to provide methodologies for either avoiding geometric errors or for proving the correctness and accuracy of algorithms dealing with geometry.

A new geometry modelling system satisfying the above requirements has been developed and implemented recently in the Monte Carlo particle transport simulation code PTSIM [15,16]. The principles of this system will be described in the following sections.

2 Surface representation of volumes

Our goal was to develop a geometry modelling system able to cover a large class of geometry bodies and combinations at the same time keeping the code simple and fast. We have opted for the surface representation modelling due to its ability to cover a wide range of volume shapes using a library of surfaces and a limited set of rules, as opposed to CSG modelling that requires the implementation of a comprehensive collection of geometrical bodies.

Mathematically a surface S can be described as the set of points $\vec{r} \equiv (x, y, z)$ for which $f(\vec{r}) = 0$. In order to distinguish between the space regions separated by a surface we will use the following sign convention. The positive space region is that for which $f(\vec{r}) > 0$, the negative region is that one with $f(\vec{r}) < 0$; of course, for the points belonging to the surface, $f(\vec{r}) = 0$. With this convention, the logical position of a point $\vec{r}$ with respect to a surface S will be characterized by the following sign function $\text{sgn}(\vec{r}, S) = \{-1 \text{ if } f(\vec{r}) < 0,\ 0 \text{ if } f(\vec{r}) = 0,\ 1 \text{ if } f(\vec{r}) > 0\}$.

We extend the sign convention in the case of the distance from a point $\vec{r}$ to a surface S (to the closest point on the surface) by taking the distance as positive or negative depending on the value of $\text{sgn}(\vec{r}, S)$.

A volume, or more generally a space region, can be defined using its boundary

surfaces and the sign convention as the proper intersection of positive or negative space regions delimited by these surfaces. Note that only the intersection operator is used in our model, although other operators, e.g. the union or the complement operators, could be used to define volumes. Our choice allows a uniform description of all volumes, which is important for an object oriented implementation. A way to overcome certain limitations introduced by this restriction is presented further in section 5.

As an example consider two planes $P_1 : z - z_1 = 0$ and $P_2 : z - z_2 = 0$, with $z_1 < z_2$, and a cylindrical surface $C : x^2 + y^2 - R^2 = 0$. The intersection between the positive semi-space $z > z_1$ defined by the plane P_1, the negative semi-space $z < z_2$ defined by the plane P_2 and the interior (negative region) of the cylindrical surface C defines a cylinder $V = P_1^+ \cap P_2^- \cap C^-$, where the positive or negative superscripts denote the positive or the negative region, respectively.

Given the restriction to use only the intersection operator the volume V will be designated by a simplified notation

$$V = P_1 - P_2 - C \tag{1}$$

Generally, for a volume V defined by the intersection of space regions with the signs $s_i \in \{-1, 1\}$ delimited by n surfaces S_i, $i \in \{1, \ldots, n\}$, the following symbolic definition will be used:

$$V = \sum_{i=1}^{n} s_i S_i \tag{2}$$

with $s_i = \text{sgn}(\vec{r}, S_i)$, where $\vec{r} \in V$.

For convenience, when the geometry data are given in the input, unique identification numbers are assigned to each surface and volume.

3 Robust determination of the exit point of a trajectory from a volume

Consider a particle inside the volume V defined in (2). Theoretically, the intersection point of its trajectory with the volume boundary is given by the smallest non-negative distance to the intersection points of the trajectory with the volume's bounding surfaces

$$l = \min_{l_{ij} \geq 0} \{l_{ij}\} \,, \tag{3}$$

where i is the index of the surface and j refers to the j-th intersection point with the surface S_i.

For a computer application, this formulation is not sufficient, as it may result in some ambiguities due to floating point errors. For example, even if the intention was to obtain the intersection point with the boundary, the computed coordinates of the point could appear in the interior of the initial domain, or, on the contrary, in the next domain. Also, the cases when the particle is on the boundary should be judged separately, as in certain circumstances the 0-length distance to that boundary must be taken into account, while in other circumstances it must be discarded.

Various approaches were proposed to overcome this problem, e.g. by adding a small shift to particle positions or by shrinking and swelling the volumes in order to have the particle always without doubt in the proper position, or by considering that the boundaries have a certain width; or as a last resort, by discarding the trajectories found in ambiguous situations or trapped in never ending loops [2,3,6,9,17–19].

In the present model a pair of values (l, τ), where l is the distance to the intersection point and τ is the crossing sign, is used to characterize an intersection point of the trajectory with a surface. Here $\tau = 1$ when the particle passes from the negative region to the positive region, $\tau = -1$ in the opposite case, and $\tau = 0$ when the trajectory is tangent to the surface (no sign changing).

In (3) instead of using the condition $l_{ij} \geq 0$ which is subject to computations errors, we will use the topological condition of having an exit point in a more explicit way, thus giving primacy to the geometrical reasoning. For a particle with current position known to be inside the volume V, that means it is simultaneously on the s_i side of each surface S_i, an intersection point l_{ij} is accepted as candidate for exit point if it satisfies the following criteria written in algorithmic form.

if $\tau_{ij} \cdot s_i \leq 0$ **then** the point is rejected;
The intersection point is not a crossing from s_i side to $-s_i$ side of the surface S_i.
else if $l_{ij} >= 0$ **then** the point is accepted;
This is the ideal case as in equation (3).
else if $l_{ij} > \varepsilon_{\mathrm{n}}$ **then**

We may have an exit point at a negative small distance from the current position. As the particle is known to be inside, the negative distance might be due to computing errors.

if doesn't exist a point l_{ij+1} **then** the point is accepted;

The point is an exit point not followed by any reentry intersection with the surface S_i.

else if $l_{ij+1} > \varepsilon_{\mathrm{p}}$ **then** the point is accepted;

The point is an exit point followed by a reentry intersection with the surface S_i at a significant positive distance.

else the point is rejected;

Both the exit point and the next intersection are at negative distances, or the particle is very close to the reentry point that there is rather an error of calculation of l_{ij+1}.

else the point is rejected;

The intersection point is at a large negative distance significant greater than any possible roundoff errors. It must be followed by at least a reentry intersection point through the same surface.

where ε_n is a small negative number introduced for efficiency reasons. If $\varepsilon_n = -\infty$ all intersections corresponding to an exit situated at a negative distance will be tested; ε_p is a small positive number introduced to decide if the particle is placed erroneously outside the volume due to miscalculation of the next reentry intersection point l_{ij+1} rather than of the point l_{ij} as shown in Fig. 1.

The closest point from all points returned by the above algorithm is the exit point from the volume. In the case of a closed region, a regular solid, at least one is always found. In the case of an open region, for certain particle positions and moving directions, no such point will be returned; in this case an "infinite value", the largest number in the machine floating point representation, is assigned to l by convention.

By taking into consideration the above procedure, the ambiguities which may be encountered in certain limit cases are avoided. Examples of such scenarios are given below:

(1) The particle is on one of the current volume's surfaces being headed inward. This happens each time when a particle enters into a volume. In this case the null distance to the surface that the particle is on must be disregarded. This is

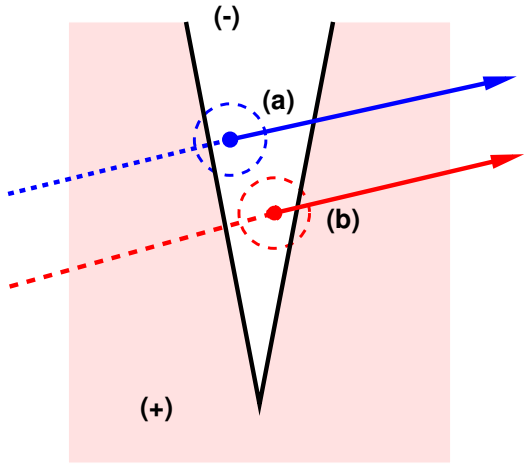


Fig. 1. A 2-dimensional representation of a case when a particle is placed outside the region where it logically belongs (the "+" region) due to floating point errors in calculation of distances to intersection points with a surface (the thick V-shaped solid line). In the case (a) the error occurs at the calculation of the distance to the exit point, while in the case (b) the error is assumed to occur at the calculation of the distance to the reentry point.

automatically done by checking the crossing signs τ_{ij}.

(2) The particle is on one of the current volume's surfaces being headed outward. This happens when a particle just left a volume. In this case that surface must be considered.

(3) The particle is on the surface with the trajectory tangent or contained into that surface. The crossing sign inequality excludes such points, avoiding thus the infinite loops which might occur otherwise, when the particle remains in the same point, but its logical position is changing from a volume to its neighbor.

(4) Due to rounding errors, the particle appears to be in the exterior of the current volume, headed inward, when in fact it should be on the boundary. This could happen when the particle just left the previous volume being transported to the separation surface, but because of the roundoff errors its coordinates are somewhere still in the previous volume. For the correct determination of the exit point from the new volume, that particular intersection point with this surface should be disregarded; this is automatically done in this model.

(5) The particle appears to be in the exterior of the current volume and is headed outward, with l found negative. In order to avoid working with negative displacements l will be taken zero. This case could happen because the magnitude of the error of the computed intersection distances may vary with the particle direction and with the type of equation which defines the surface. It

could also be caused by a shift between the solutions of the equations that give the intersection distances and the particle position with respect to the same surface. As shown in Fig. 1 the cases 4 and 5 can simultaneously occur.

An alternative way to solve the cases 4 and 5, somewhat similar to how this problem is addressed in GEANT4 [9,18–20], consists of taking the distances $l_{ij} = 0$ whenever $|l_{ij}|$ is less than a given tolerance limit and discarding the test of the next intersection point l_{ij+1}. However in narrow regions such a procedure could lead to ambiguous situations when a particle appears to be simultaneously on two distinct surfaces, sometimes causing the tracking algorithm to go into an endless loop.

4 Determination of the next volume the particle passes

Denote by λ the free distance to the next discrete interaction in the volume V_i. If $l < \lambda$ then the particle should be transported to the boundary of the volume, $\vec{r}_s = \vec{r}_0 + l \cdot \vec{u}$. Suppose that S_j is the exit surface. Once the particle reaches this point the adjacent volume into which the particle passes must be determined. In order to be able to do this, for each surface S_j of a volume V_i the list with the adjacent volumes $\{V_{k_{ij}}\}$, $k_{ij} \in \{1, \ldots, n_{ij}\}$, separated by the surface S_j is constructed (see below). This is done in a preliminary step, immediately after the input of geometry data.

The identification of the volume into which the particle passes can be done by testing successively in which volume from the list $\{V_{k_{ij}}\}$ the particle position $\vec{r}_s$ belongs. If no volume is found then it is assumed that the particle leaves the domain of interest. The procedure for testing the particle position with respect to a volume will be presented later in section 5.1.

The list $\{V_{k_{ij}}\}$ of the volumes adjacent through the surface S_j to the volume V_i, can be built automatically by searching the volumes that have a common surface and are placed on its opposite sides. Note that besides the volumes in contact, which are in fact the only ones of interest, this procedure can add to the list other volumes that are not adjacent. The only drawback of using an enlarged $\{V_{k_{ij}}\}$ list is the fact that an additional number of unnecessary comparisons could be made for selecting the next volume. The inclusion of non-adjacent volumes in the list can be avoided by giving to the user the possibility to specify for each volume a list where to search

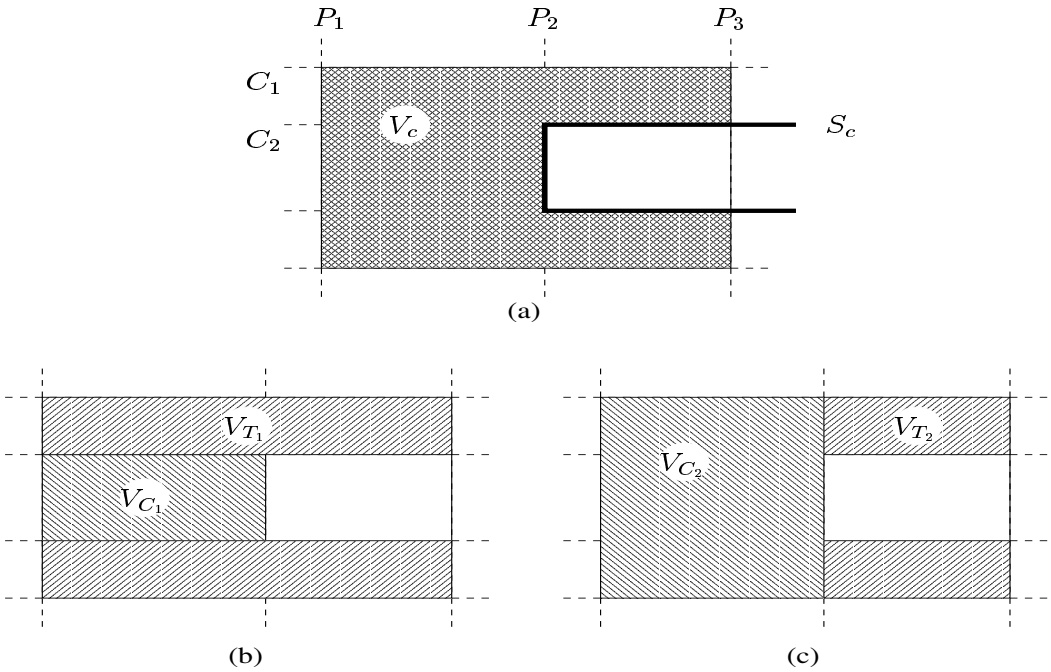


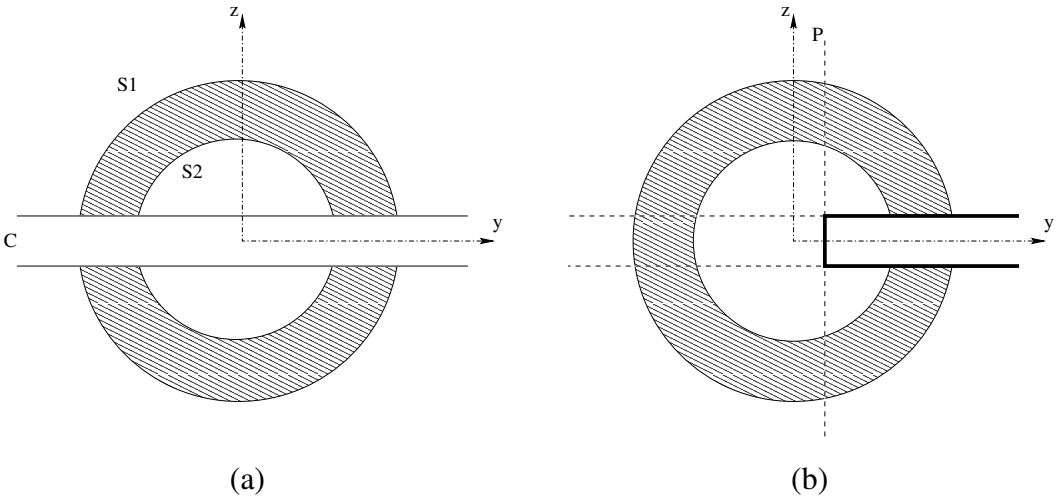
Fig. 2. Example of using a compound surface for defining a volume with a concavity.

for adjacent volumes or the volumes that must be excluded.

5 Compound surfaces

Using only intersections of space regions delimited by elementary surfaces to define arbitrary volumes may prove to be a very laborious task requiring the artificial decomposition of the domain into too many volumes. Such difficulties occur in the case of volumes with concave shapes defined by distinct surfaces. As an example consider (Fig. 2) three planes $P_1 : z - z_1 = 0$, $P_2 : z - z_2 = 0$ and $P_3 : z - z_3 = 0$ with $z_1 < z_2 < z_3$ and two cylindrical surfaces $C_1 : x^2 + y^2 - R_1^2 = 0$, respectively $C_2 : x^2 + y^2 - R_2^2 = 0$ with $R_1 > R_2 > 0$.

Using these elementary surfaces two cylinders, $V_1 = P_1 - P_3 - C_1$, and $V_2 = P_2 - P_3 - C_2$ can be defined. But the volume $V_c = V_1 \setminus V_2$ (see Fig. 2a) cannot be defined using only the elementary surfaces defined above. This volume can be represented by dividing it in two other volumes, e.g. using the tube $V_{T_1} = P_1 - P_3 - C_1 + C_2$ and the cylinder $V_{C_1} = P_1 - P_2 - C_2$ (Fig. 2b), or using the cylinder $V_{C_2} = P_1 - P_2 - C_1$ and the tube $V_{T_2} = P_2 - P_3 - C_1 + C_2$ (Fig. 2c).



(a) (b)

Fig. 3. Example of using a compound surface for avoiding ambiguities in volume definition.

A way to avoid this kind of additional volume decomposition is to use *compound surfaces*. Given a volume, or an open space region, a compound surface is defined as its boundary surface taken as a whole. Formally it is defined as the union of the intersections between each boundary surface and the given space region.

In the above example we can define the compound surface S_c as the surface of $P_2 - C_2$ (see Fig. 2a). For the compound surfaces we use the sign convention (described in more detail in 5.1) that the positive sign corresponds to the region interior to the compound surface, so that the compact notation $Q - S_c$ represents the intersection of some region Q with the region exterior to the S_c. This notation allows us to express the volume V_c simply as $V_c = P_1 - C_1 - P_3 - S_c$.

Compound surfaces prove to be useful also for eliminating certain ambiguities in defining the space regions. As an example, a volume incompletely defined only by its border surfaces is presented in Fig. 3; this example was discussed also in [4] where a concept of ambiguity surface is used. Consider two spheres S_1, S_2 centered on the origin and a cylindrical surface C around the y axis. Suppose that the volume V represented by the hatched region in Fig. 3b should be defined.

If we try to define it as the intersection of the interior of the sphere S_1 with the exterior of the sphere S_2 and the exterior of the cylindrical surface C, that means $V = -S_1 + S_2 + C$, we obtain the space region formed by the two hatched regions in Fig. 3a, $W = -S_1 + S_2 + C$, which is not the desired volume.

Instead of using directly the cylinder C we can introduce an additional plane P

as in Fig. 3b and take the surface S_c, marked with thick line, defined by the space region formed by the semi-space at the right side of the plane P intersected with the interior of the cylindrical surface C, $S_c =$ surface of $P - C$. Then the desired volume can be correctly defined as $V = -S_1 + S_2 - S_c$.

In order to solve geometrical problems (position relative to the surface, intersection points, distance to the boundary) using compound surfaces in a transparent and simple way, like in the case of elementary surfaces, the procedures developed for elementary surfaces should be extended to the case of compound surfaces.

5.1 Determination of the position sign of a point with respect to a compound surface

The position sign of a point with respect to a compound surface is defined as: 0 if the point is situated on that surface (on one of its components), 1 if it is in the interior of the region delimited from the rest of the space by that surface, and -1 if it is in exterior. Consider a space region $\sum_{i=1}^{n} s_i S_i$ where $s_i \in \{-1, 1\}$ and denote by S_c the corresponding compound surface. Accordingly, the algorithm for obtaining the position sign with respect to a compound surface is the following:

If there exists a surface S_i for which $\text{sgn}(\vec{r}, S_i) \cdot s_i < 0$ then $\text{sgn}(\vec{r}, S_c) = -1$, otherwise if the point is on one of the surfaces S_i then $\text{sgn}(\vec{r}, S_c) = 0$, else $\text{sgn}(\vec{r}, S_c) = 1$.

In this case also the ambiguities which may result due to finite precision computations must be avoided. For example, when the particle passes from one volume to another, logically its position is situated on the separation surface between the two volumes, but its new coordinates could be somewhere still in the interior of the first volume. For this reason, when testing the particle position with respect to an adjacent volume, the test against the common surface is skipped.

5.2 Determination of the intersection points of a trajectory with a compound surface

Consider a compound surface defined as above and a particle in the position $\vec{r}_0$ with moving direction $\vec{u}$. We are interested in finding the intersection points of the particle's trajectory with the compound surface. This can be done using the following algorithm.

(1) Search successively to find the intersection points with each surface S_i, $i \in \{1, \ldots, n\}$.
(2) When for a certain surface S_i such a point is found, at distance l_{ij} from $\vec{r}_0$, test if it belongs to the region delimited by the remaining surfaces S_k, $k \neq i$.
(3) If there exists a surface for which the point $\vec{r}_0 + l_{ij}\vec{u}$ is outside, then the point is not a valid intersection with the whole compound surface. Otherwise this point is one of the points we are looking for. Its crossing sign is $\tau_{ij} \cdot s_i$.
(4) Repeat the procedure from the first step until all points on all surfaces are tested.

The test of the position of an intersection point l_{ij} with respect to a surface S_k needed in step 2 can be done in two ways.

First it can be done by computing the coordinates of the point $\vec{r} = \vec{r}_0 + l_{ij} \cdot \vec{u}$ and testing its position with respect to these surfaces. This procedure introduces the problem of producing topologically consistent lists of intersection points. Due to rounding errors at edges correct intersection points can be rejected, or points outside the domain could be accepted. The problem of rejected points can be corrected by accepting the points within a given tolerance distance from the bounding surfaces. The additional erroneous introduced points can be eliminated by screening the list for breaks in the alternation of crossing signs of successive points, $\tau_m \cdot \tau_{m+1} \leq 0$.

Another method, more consistent, exploits the geometrical meaning of the order relations between the intersection points. Let l_{km} be the intersection with the surface S_k that is nearest to the point l_{ij}. The point is inside if $s_k \tau_{km}(l_{ij} - l_{km}) \geq 0$, else it is outside. If we have no intersection with the surface S_k, then the position sign of the initial point $\vec{r}_0$ is relevant; the point is inside if $\mathrm{sgn}(\vec{r}, S_k) \cdot s_k \geq 0$.

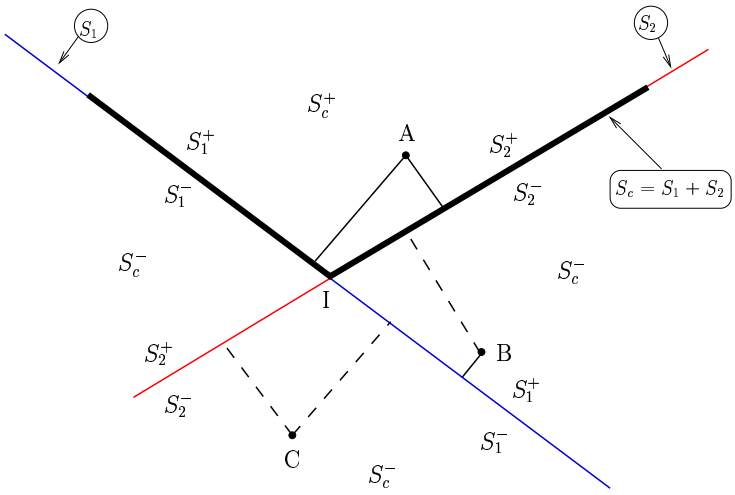


Fig. 4. Determination of the distance from a point to a compound surface. The positive distances are represented with solid lines while the negative distances are represented with dashed lines.

5.3 Determination of the distance from a point to a compound surface

Keeping the sign convention used in the definition of the distance from a point to an elementary surface, in the case of compound surfaces (and of volumes) the distance from a point $\vec{r}$ to a surface S_i of the volume V is $d_i = d_i' \cdot s_i$, where d_i' is the signed distance to S_i according to the definition in the third paragraph of section 2.

If the point $\vec{r}$ is in the interior of V where $\text{sgn}(\vec{r}, S_c) > 0$ (the case of point A in Fig. 4) then the distance we are looking for is the smallest among all the distances to the surfaces S_i, $i \in \{1, \ldots, n\}$, each being nonnegative. If the point $\vec{r}$ is outside V where $\text{sgn}(\vec{r}, S_c) < 0$ (like the points B and C in Fig. 4) then we can have both positive and negative distances to the elementary surfaces S_i. In this case the only significant distances are the distances to the surfaces with respect to which $\vec{r}$ is in exterior, that means only the negative distances. In consequence the distance we are looking for is the greatest in absolute value from all the distances $d_i < 0$. Thus we can apply an unique rule: the smallest of all distances regardless of their sign.

As can be seen from the case of point C in Fig. 4, when the distance to the compound surface is selected from negative distances, the result may lose the significance of the distance to the closest point on the surface (in our case the point I). This results in an underestimate of the real distance, more manifest at large distances. However, the value returned by the above algorithm is still useful in most

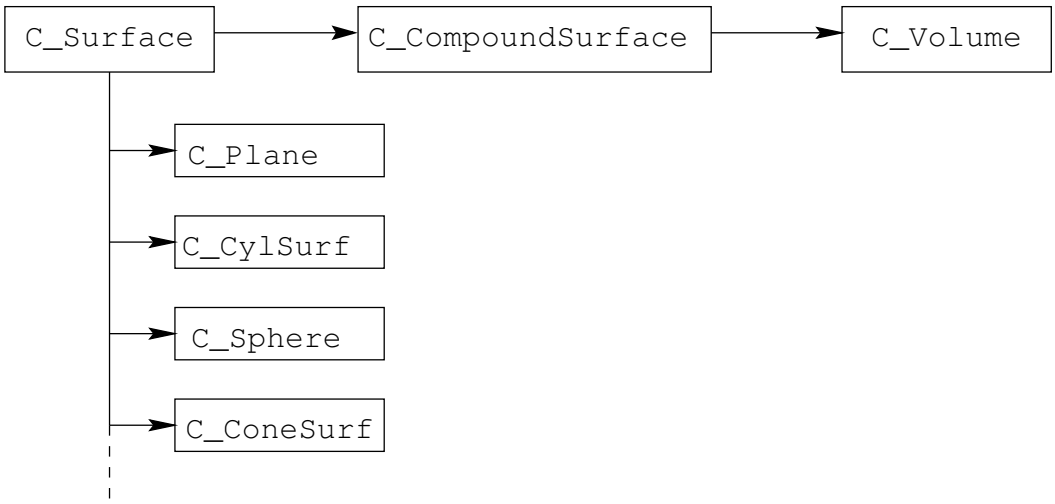


Fig. 5. Schematic representation of the class hierarchy used in the implementation of the geometry modeling system proposed.

practical cases when the proximity of the boundary must be detected in order to trigger certain actions like change of the step size selection method for particles with curved trajectories, or tallying of certain quantities of interest.

6 Notes about implementation

The geometry modelling system was implemented using object oriented techniques. It was coded using C++ programming language. The code is structured into a class hierarchy as described in Fig. 5.

The class `C_Surface` is defined as an abstract class, from it being derived all other classes implementing specific surface types. Its main methods are `GetPositionSign`, `GetIntersections`, `GetNextIntersection` and `GetDistance`, used to implement the functions described in sections 2 and 3. In the case of class `C_CompoundSurface` the above methods implement the algorithms described in section 5 which are inherited also by the class `C_Volume`. In addition the class `C_Volume` has methods for dealing with the particle passage to adjacent volumes as described in section 4.

For initialization first the instances of specific surface types (planes, cylindrical surfaces, ...) are created; then, if it is the case, the compound surfaces, and finally

the volumes (a volume once declared can be used also as a compound surface). At creation time each instance receives an unique identification number. After creation of all volumes the routines for the construction of the lists with adjacent volumes as described in section 4 are called.

7 Conclusion

The principles of a geometry modelling system for particle or ray tracking simulation are presented. It uses the surface representation modelling offering flexibility in describing a wide range of volume shapes rather than combining a limited set of predefined geometrical bodies as in the case of Constructive Solid Geometry.

The particle tracking algorithm uses in a systematic manner additional information about the topological description of the particle position and trajectory with respect to cells surfaces in order to accurately determine position and intersection points, avoiding ambiguities induced by floating point rounding errors.

The basically uniform definition of volumes, using only the intersection operator, allows the possibility to create complex structures by using compound surfaces, and at the same time leads to an efficient and natural implementation using object oriented programming.

Acknowledgments

The author expresses his gratitude to Prof. Robert Lewitt and Prof. Octavian Sima for their suggestions and help in revising the manuscript. The author is also grateful to Prof. Mircea Oncescu for his support and encouragement in carrying out this work.

References

[1] W. R. Nelson, H. Hirayama, D. W. O. Rogers, The EGS4 code system, SLAC Report 265, Stanford Linear Accelerator Center, Stanford University (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, CERN (Nov. 1980).

[3] W. R. Nelson, T. M. Jenkins, Geometry methods and packages, in: T. M. Jenkins, W. R. Nelson, A. Rindi (Eds.), Monte Carlo Transport of Electrons and Photons, Plenum Publishing Co., 1988, Ch. 17, pp. 385–405.

[4] J. F. Briesmeister, MCNP4 – a general Monte Carlo code for neutron, photon and electron transport, Report LANL 7396-M, Rev. 4, LANL (1991).

[5] R. Brun et al., GEANT Detector description and simulation tool, Report W5013, CERN Program Library (1994).

[6] P. A. Aarnio, Particle tracking across boundaries in simulation of high energy hadronic and electromagnetic cascades, Physica Scripta T33 (1990) 147–151.

[7] P. Andreo, M. Ljungberg, General Monte Carlo codes for use in medical physics, in: M. Ljungberg, S.-E. Strand, M. A. King (Eds.), Monte Carlo Calculations in Nuclear Medicine, Applications in Diagnostic Imaging, Institute of Physics, Bristol and Philadelphia, 1998, pp. 37–52.

[8] O. Sato, S. Iwai, M. Nakamura, T. Uehara, S. Takagi, H. Hirayama, UCMARS — a user code with a multiple-array system using combinatorial geometry for EGS4, Report KEK 94-12, National Laboratory of High Energy Physics, Japan (1994).

[9] J. Apostolakis, RD44 collaboration, An overview of GEANT4 geometry, in: International Conference on Computing in High Energy Physics, CHEP 97, Berlin, 1997.

[10] S. Fortune, Robustness issues in geometric algorithms, in: Lin and Manocha [21], pp. 9–14.

[11] L. J. Guibas, Implementing geometric algorithms robustly, in: Lin and Manocha [21], pp. 15–22.

[12] C. M. Hoffman, J. E. Hopcroft, M. S. Karasick, Towards implementing robust geometric computations, in: Proceedings of 4-th Annual ACM Symposium on Computational Geometry, 1988, pp. 106–117.

[13] C. M. Hoffman, Robustness in geometric computations, Journal of Computing and Information Science in Engineering 1 (2001) 143.

[14] C. Yap, Towards exact geometric computation, Computational Geometry: Theory and Applications 7 (1997) 3–23.

[15] L. M. Popescu, Monte Carlo simulation of radiation transport through substance, Ph.D. thesis, Atomic Physics Institute, Bucharest, in Romanian (Nov. 1999).

[16] L. M. Popescu, A computer code package for Monte Carlo electron-photon transport simulation. Comparisons with experimental benchmarks, Nuclear Instruments and Methods in Physics Research B 161–163 (2000) 318–322.

[17] A. F. Bielajew, HOWFAR and HOWNEAR: Geometry modelling for Monte Carlo particle transport, Tech. Rep. PIRS–0341, National Research Council of Canada (1995).

[18] P. Kent, Minimising precision problems in GEANT4 geometry, GEANT4 working note, citation in [9] (Apr. 1995).

[19] P. Kent, Pure tracking and geometry in GEANT4, GEANT4 working note, citation in [9] (Apr. 1995).

[20] D. C. Williams, GEANT4 geometry/tracking questions, notes posted on the author's web page at Santa Cruz Institute for Particle Physics.
URL `http://scipp.ucsc.edu/~davidw/geant4/geant4.html`

[21] M. C. Lin, D. Manocha (Eds.), Applied computational geometry: towards geometric engineering; selected papers / FCRC '96 Workshop, WACG '96, Springer Verlag, Philadelphia, PA, 1996.