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

XPERT – A PACKAGE FOR GEOMETRY DEFINITION, VERIFICATION AND VISUALISATION

James Tickner[*]
CSIRO Minerals
PMB 5, Menai, NSW 2234, Australia
James.tickner@csiro.au

ABSTRACT

The task of defining, verifying and maintaining a complex, 3-dimensional geometry model for a modern Monte Carlo code typically occupies a large fraction of the total time required to set up a particle transport simulation. To address this problem, the computer code XPERT has been developed. Key features of this system discussed in this paper include: a constraint-based, hierarchical, constructive solid geometry modeler; efficient, automated geometry validation routines and real time, 3-dimensional wire-frame and solid geometry visualization. The XPERT system has been interfaced to the EGS and MCNP codes as well as an in-house, optical photon transport code. Examples illustrating the above features are presented for these codes.

Key Words: Monte Carlo, geometry modeling, CSG, GUI

1 INTRODUCTION

One of the major advantages of the Monte Carlo method for solving transport problems is that it can support, in principle, an arbitrarily detailed geometry model. As computer speeds and memory capacities have increased, so has the complexity of simulated systems. Models of large, high-energy physics detectors such as ATLAS may contain in excess of 10^7 separate detector and passive volumes [1], albeit with considerable numbers of repeated structural elements. Even bench-scale nuclear instruments may comprise hundreds of parts that must be modeled in an accurate simulation.

The On-line Analysis and Control group of CSIRO Minerals develops instrumentation for the minerals and security industries using X-ray and beta-ray transmission, X-ray fluorescence (XRF), X-ray diffraction (XRD) and neutron/gamma technologies in addition to a wide range of non-nuclear methods. The XPERT system was developed to support the application of Monte Carlo methods to the design, optimization, calibration and analysis of a wide range of nuclear instrumentation. In particular, the code provides tools for:

- Rapidly specifying a problem geometry using a constraint based, hierarchical, constructive solid geometry modeler;
- Facilitating modification and optimization of the geometry model;
- Automatic error checking;

[*] Tel. +61 2 9710 6732. Fax. +61 2 9710 6789

- Converting the geometry into a boundary representation (BRep) model allowing it to be displayed and manipulated in real-time and exported to computer aided design (CAD) packages; and
- Overlaying particle tracks and tally results onto the geometry model.

The XPERT code has been interfaced to EGSnrc [2], MCNP [3] and an in-house optical photon transport program. It is written primarily in the TCL/TK [4] scripting language that is supported on a wide range of platforms.

2 GEOMETRY MODELS FOR PARTICLE TRANSPORT

2.1 Background

The majority of particle transport Monte Carlo codes assume that the geometry being modeled is piecewise homogenous and consists of a finite number of regions each of which has uniform composition and density. Given a specified point and direction vector, the geometry modeler has to be able to answer the following questions:

Q1. Which region does the point lie in?

Q2. How far away is the nearest boundary in the specified direction of travel and which region lies across the boundary[1]?

Q3. What is the shortest distance to any boundary of the region containing the point?

The two main approaches to specifying a geometry model are the surface and constructive solid geometry (CSG) representations.

2.2 Surface representation

In the surface representation, each region is described by specifying the sense of points within the region with respect to one or more surfaces. A common simplification is to only allow surface equations up to second order. Answering Q1 then just requires evaluation of the signs of the quadratic surface functions F and checking which region definition is satisfied. The solution to Q2 requires the solution of a linear or quadratic equation to calculate the distance to each surface bounding the region containing the selected point. The shortest distance which leads to a point on the boundary of the region is selected. Curiously, answering Q3 for a general quadratic surface is surprisingly hard, requiring the solution of a degree-6 polynomial. However, for linear (planar) surfaces and a variety of specific quadratic surfaces, notably cylinders, spheres and cones, the problem again reduces to solving a quadratic equation. Round-off errors due to the finite precision of a computer's floating-point number representation can lead to problems if these results are applied naively, but overcoming these is straightforward [6].

Suppose a surface i is defined by the equation $F_i(x,y,z)=0$. The surface divides space into two regions, with $F_i>0$ and $F_i<0$; let r_i and $\bar{r}_i$ respectively denote these two regions.

[1] The Woodcock [5] or delta-tracking algorithm dispenses with the need to calculate boundary distances, at the cost of increased inefficiency if materials with a wide range of cross-sections are present.

A general volume V can then be written as

$$V = \sum_{l} \left[\prod_{m} r_{lm} \prod_{n} \bar{r}_{ln} \right] \tag{1}$$

where the product of two regions denotes their intersection and summation denotes union.

The surface representation is extremely flexible, allowing any volume bounded by the family of permitted surface functions to be described. It is also a natural representation for implementing as a computer code to answer the three tracking questions. However, it requires considerable work by the user. For example, to specify a volume as simple as a 3-dimensional rectangular box, the user must calculate the equations of the 6 planes bounding the box and the sense of points inside the box with respect to these planes.

2.3 Constructive solid geometry (CSG) representation

The CSG or combinatorial geometry (CG) representation considerably simplifies the task of specifying a geometry model. The basic element of the CSG representation is a volume primitive. A primitive is one of a family of simple shapes whose size, position and orientation are specified by the user. Common primitives include 3-dimensional boxes, cylinders, spheres and prisms. A general volume is then defined by combining one or more primitives using Boolean union, intersection and difference operators. Provided that the suite of volume primitives matches the range of shapes required in a particular problem, the CSG representation can provide a simple and intuitive tool for geometry definition. The most elegant method for implementing the CSG method into a computer code is to perform an initial preprocessing step where the CSG model is converted into an equivalent surface model. Q1-Q3 can then be answered straightforwardly using the results described in section 2.2.

One of the drawbacks of both the surface and CSG geometry models is that all surfaces and volumes are defined in a universal coordinate system. Whilst this is convenient for the implementation of particle tracking, it can be inconvenient for the user. For example, if a problem contains widely separated, finely detailed regions, it is necessary for the user to calculate large offsets when describing the positions of the detailed volumes. If the user subsequently decides that one of the detailed regions (a particle detector, for example) needs to be moved, then it is necessary to rewrite the description of every volume or surface forming that component. The user must also make sure that every point lies in one and only one volume to ensure that the geometry is unambiguously specified.

These difficulties can be avoided by using a hierarchical geometry model [7]. The model is constructed of volume primitives positioned relative to a “parent” volume. The first primitive in the problem is positioned inside an infinite void. All particles entering the void are discarded. Volumes may also contain “child” volumes. When a child volume is positioned inside its parent, a hole is automatically made for it to ensure that only one volume is defined at each point in space. The main advantages of the hierarchical model are: enforcement of a logical structuring of the problem geometry; simplification of the description of detailed regions because all volumes are positioned relative to a local parent volume; easy duplication of parts of the model by simple duplication of parent volumes; and facilitation of easy modifications to the model. As with the regular or absolute CSG representation, the hierarchical geometry model is best implemented via a pre-processing stage to convert locally specified volumes into globally specified surfaces.

2.4 The Constraint-Based, Hierarchical CSG model

Volume primitives in either absolute or hierarchical geometry models are typically positioned with respect to a fixed point, usually their centre. Their dimensions are also usually given in absolute terms. This introduces two related problems. Firstly, specifying volume positions in this way is non-intuitive and can involve considerable time-consuming arithmetic for the user, even for simple cases where volume faces are aligned with the coordinate axes. For problems with arbitrarily rotated volumes, the calculation of volume positions rapidly becomes tedious. Secondly, maintaining and modifying the geometry model is unnecessarily hard, due to the way that volume positions and dimensions are interrelated. Changing the size or position of a single volume can require the user to manually propagate changes through large parts of the model.

The constraint-based, hierarchical CSG model has been developed to overcome these difficulties. It allows users to set up a geometry model in a way that more naturally corresponds with the way that an object is physically constructed. The model is fundamentally a hierarchical CSG system as described in section 2.3. The unique feature of the model is the way in which volumes are positioned and dimensioned by placing constraints on points lying on the surface or within the volume being defined.

Each volume primitive is positioned inside a rectangular *bounding box* that is just large enough to enclose the primitive. Currently supported primitives include boxes (rectangular parallelepipeds), spheres, right circular cylinders, frusta and right 3- or 4-sided prisms. The bounding box serves to define both the position and dimensions for box, sphere, and cylinder primitives. Frusta and prism primitives require additional parameters to define their maximum and minimum radii and end-face coordinates respectively.

Figure 1 shows a cylinder primitive inside its bounding box. Twenty-seven constraint points are defined for each bounding box, positioned at each vertex (8 points), the centre of each edge (12 points), the centre of each face (6 points) and the centre of the box (1 point). These constraint points are shown as red, green and blue spheres. Bounding boxes have their edges parallel to the coordinate axes and can be defined by fixing the positions of a minimum of two and a maximum of 4 of their constraint points. One point must be fixed relative to a constraint point of another, previously defined volume[2]. Subsequent points can either be similarly fixed, or they can be defined relative to a point in the same volume. The latter method defines an absolute size for the volume being defined.

Primitives can be arbitrarily rotated about any point of their bounding box. If one primitive is constrained relative to another rotated primitive, an additional displacement is automatically calculated such that the two primitives satisfy the same constraints at their bounding boxes. For example, if two bounding boxes are defined to have two coplanar faces, then the displacement factor ensures that the two rotated primitives also have coplanar faces. This allows complex, arbitrarily rotated volumes to be constructed with minimal effort.

[2] The initial, infinite void volume has the origin as its sole constraint point.

In addition to the implicit difference operation where holes are automatically made in a parent volume to accommodate its children, two explicit Boolean operations are also implemented. Firstly, an intersection operation allows a child volume to be explicitly clipped by its parent. (Normally it is the responsibility of the user to ensure that a child fits inside its parent). Secondly, an explicit difference operation is included, which allows a single volume to pass through several "parents" – for example, a beam-pipe passing through a complex detector. Together, these operations increase the flexibility and ease of use of the model and allow a greater range of volume shapes to be defined.

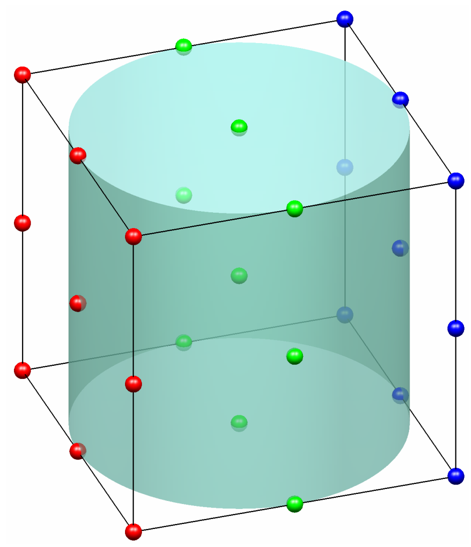

Figure 1. A cylinder primitive inside its bounding box. Constraint points are shown as coloured spheres.

A key part of the constraint-based model is that all bounding boxes are recalculated whenever any part of the geometry is changed. This means that changes made to one part of the geometry automatically propagate through the whole model. A logical top-down ordering of volumes is also imposed. Significant changes can be made by changing the positions or dimensions of a small number of volumes, whilst preserving the logical consistency of the geometry model.

2.5 Implementation

The constraint-based geometry model has been implemented into a graphical user interface written in the TCL/TK scripting language. Primitives and their bounding boxes can be viewed in 3-dimensions from any orientation and at varying levels of detail as the model is constructed.

New volumes are defined by selecting a primitive type and placing constraints on its bounding box. The 27 constraint points for each volume have a 3 digit identifier, with digit values of 1, 2 or 3 indicating low, central of high values for the X, Y and Z axes. For example, point P_{222} is the centre of the bounding box. A constraint of form $P_{ijk}(VOL1)=P_{lmn}(VOL2)+(\Delta x, \Delta y, \Delta z)$ indicates that the position of constraint point P_{ijk} for volume *VOL1* is fixed to the position of point P_{lmn} of volume *VOL2* plus the specified offset. If *VOL1=VOL2* then the constraint provides an absolute specification of one or more dimensions of volume *VOL1*. Constraints can be constructed by either manually entering the constraint points and offsets or by clicking on points shown in the graphical display.

The constraint list is then processed as follows. For each coordinate axis, the bounding box half-edge-length *A(b)* and the coordinates of the low, central and high bounding box points *A(1,2,3)* where *A=X, Y* or *Z* are initialized to “unset”. A pass is then made through all of the constraints. For absolute (*VOL1=VOL2*) constraints, the appropriate bounding box dimension is set if the two constraint point indices differ – for example, the *Y* dimension of the bounding box is set if $j \neq m$. For relative (*VOL1 ≠ VOL2*) constraints, the low, central or high bounding box position is set. A second pass is then made. Any missing *A* values are calculated from the relations *A(3)=A(2)+A(b)=A(1)+2A(b).* This algorithm has three possible outcomes: all *A* values are set, in which case the bounding box is fully specified; a conflict arises in which an already defined *A* value is recalculated to have a different value, in which case the constraints are incompatible and an error is raised; or some of the *A* values remain unset, in which case further constraints are required. The constraint list is processed each time a constraint is added or amended. Once the bounding box is fully specified, no more constraints can be added.

Volumes can also be defined via a geometry macro. As an interpreted language, TCL/TK can execute text strings directly as scripts. By providing routines to interrogate and manipulate the geometry model, users can write simple TCL/TK scripts to add new volumes. Because the full TCL/TK language is available including loops, conditional statements and mathematical operators, the user can rapidly create very complex models including, for example, arbitrary nested or repeated structures or complicated constraints. Regular and macro-defined volumes can be freely mixed and the results viewed and checked at any stage.

Whenever a volume definition is altered, the constraint lists for all other volumes that reference this volume are reprocessed. If the bounding boxes of any of these volumes change, the process is repeated recursively. The geometry macro is also rerun. In this way, changes to a single volume propagate through the entire model.

For the purposes of particle transport, the volume model is converted to a surface representation. The equations of the planar, cylindrical, spherical and conic surfaces required to bound every volume are calculated. Care is taken to ensure that surfaces that are identical within a specified tolerance are forced to match exactly to avoid subsequent tracking problems due to numerical round-off errors. Signs are arranged such that the region inside a volume corresponds to the positive side of the surface. Using the notation of equation (1), the definition of volume V_j can be written as

$$V_j = \prod_i r_{ij} \prod_m \left[\sum_n \bar{r}_{mn} \right] \tag{2}$$

where r_{ij} denotes the region associated with the *j*th surface of volume primitive *i*, the product over *m* extends over the volumes that have *j* as a parent or are explicitly differenced from *j* and the sum over *n* extends over the *n* surfaces of volume *m*. If volume *j* is explicitly bounded by its parent volume, then the product over *i* should be extended to include the surfaces of the parent volume as well as the surfaces of *j*. Once it has been calculated on which side of each surface a particular point lies, it is straightforward to apply equation (2) to work out in which volume the point lies.

3 AUTOMATED GEOMETRY VALIDATION

It is critical that a geometry model is consistent – that is, that every point lies in one and only one volume. If this is not true, then particle tracking has to be terminated when a particle reaches a volume where it is not possible to unambiguously assign a material. The constraint-based geometry system and the ability to interactively view the model being constructed allow most mistakes to be readily detected, but fully checking a complicated model can be time consuming.

Using the constraint-based system, the most common mistake (type 1) is positioning a volume inside the wrong parent volume. For example, if volume A contains volume B which in turn contains volume C, users will frequently declare both volume B and C as children of A. This leads to an error where B and C intersect. The other common error (type 2) is to misplace one volume so that is overlaps another.

The simplest, naïve method for finding geometry errors is to randomly sample points within a volume large enough to include all the volumes defined in the problem. A check is made against each volume to see whether it includes the sampled point. If exactly one volume includes the point, a new point is sampled; otherwise the algorithm halts and reports the error. If no points are found in error, the algorithm is terminated after a specified number of sample points or time interval. This algorithm (ALG1) is straightforward but inefficient if the erroneous region occupies only a small fraction of the total volume.

An improvement [3, pg 3-8] is to sample *tracks* rather than points through the problem geometry. A track is started at a random point sampled either within the geometry model or on a surface large enough to enclose the entire model. A random direction is chosen and the track is followed until it reaches the void. The correctness of the geometry is checked where the track intersects each surface defined in the model.

Both ALG1 and ALG2 efficiently find large erroneous volume but require many samples to find small, problematic regions. Users however are just as likely to make mistakes defining small volumes as defining large ones. Biasing the sampling of the checked or track starting points to reflect this leads to a significant efficiency gain. A point is randomly sampled within one of the defined volumes, which is itself randomly chosen with equal probability for each volume. A check is made of the number of volumes at the selected point (ALG3) or along a track heading in a randomly selected direction from the sampled point to the void (ALG4).

To investigate the efficiency of these 4 algorithms, 2 test problems were evaluated. In the first, a 1-unit cube is positioned inside a 10-unit cube, with both volumes having the same parent. In the second problem, two 1-unit cubes, overlapping each other by 0.01 units are both positioned inside a 10-unit cube which is their parent volume. The average number of trials to find the erroneous region is reported in table 1 for the two test problems and 4 algorithms.

Table I. Number of trials required to find geometry error for 2 test cases for track sampling algorithms ALG1-ALG4

Algorithm	ALG1	ALG2	ALG3	ALG4
Case 1	1000	60.7	2.00	1.50
Case 2	1×10^5	308	147	8.16

Algorithm 1 is particularly inefficient due to the small fraction of the total volume occupied by the erroneous region. Algorithms 2-4 require increasingly fewer samples to find the error. The two track-based algorithms ALG2 and ALG4 take approximately 4-5 times longer to execute per sample than ALG1 and ALG3. This makes ALG3 the most efficient for finding the easily detected type 1 errors, but ALG4 is significantly more efficient for finding the more difficult type 2 errors, particularly when the degree of overlap is small. ALG4 is implemented in the XPERT code. The user can initiate a geometry check at any stage. If an error is found, a 2-dimensional cut through the geometry model is displayed, centered on the erroneous point, along with a list of volumes found at that point. The user can then locate and correct the error.

4 THREE DIMENSIONAL GEOMETRY, TRACK AND TALLY DISPLAY

4.1 Display of 3-dimensional geometry models

A CSG based model is convenient for defining a Monte Carlo problem geometry whereas the surface representation, based on the intersection of regions defined by infinite planar and quadratic surfaces, is the most suitable for particle transport. The conversion from the former to the latter is straightforward as described in section 2. However, neither of these representations is well suited to the real-time display of a 3-dimensional geometry model.

Three-dimensional computer graphics APIs such as OpenGL [8] require a geometry model to be specified using finite, planar polygons, the so-called boundary representation or BRep model. These polygons may not contain holes, so surfaces with holes must be constructed from unions of simpler polygons. Box and prism primitives can be represented exactly; primitives with curved surfaces can be approximated using an appropriate number of flat polygons.

One approach that avoids the need to explicitly calculate the surface polygons of complex volumes is *image-space* CSG rendering, for which numerous algorithms have been proposed [9-12]. These use the depth and stencil buffers provided in modern computer graphics hardware to build up a 3-dimensional image by only displaying the parts of the each primitive's surface that survive the CSG operations. This approach is appealing because it works directly from the CSG representation of the geometry model. However, because the operations are performed in image rather than object space, they need to be performed each time the viewing position, direction or scale is changed. Consequently, for all but the simplest models, the display update rate is too slow.

An alternative algorithm has been developed that calculates an explicit polygon representation of a CSG volume. It takes the finite polygon models of the primitives that comprise the volume, splits polygons that lie partly inside and partly outside the volume and then discards all polygons that lie outside of the CSG volume. In more detail the algorithm comprises the following steps:

(a) For each volume, prepare a polygon representation of the volume primitive, its mother primitive (if it is explicitly bounded) and any child or difference primitives. Only triangles are used, so quadrilateral faces have to be split into 2 triangular pieces.

(b) Identify every pair of intersecting triangles. Triangles that are intersected are split into 2 or 3 triangular pieces as shown. This process is recursive – that is, triangles produced by splitting are themselves checked for intersections and may be split further. At the end of this step, the model consists of triangles, all of which lie entirely inside or outside of the final CSG shape.

(c) Identify triangles (here coloured red) that lie outside of the final CSG shape

(d) Remove these triangles from the model

(e) Locate all 'T' intersections. T intersections occur when two triangles share a vertex that lies on the edge of another triangle and they lead to ugly cracks or gaps when rendering the final image. Remove all T intersections by further triangle splitting (shown in green)

(f) When producing a wire frame display, remove all internal triangle edges. An edge is internal if it is shared by two triangles, both of which have the same normal vector.

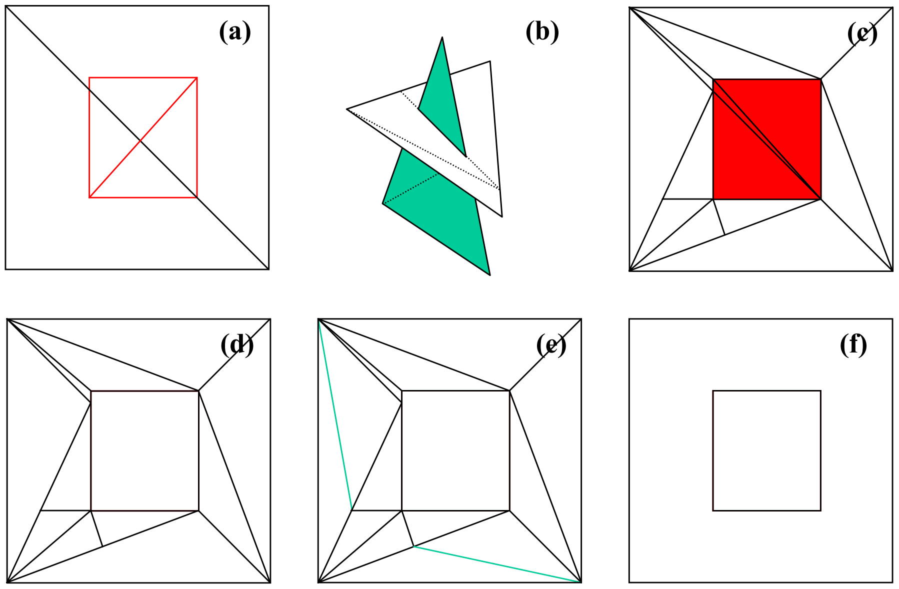

Figure 2. Illustration of steps required to prepare an explicit polygon surface representation of a CSG volume. See text for details.

After the triangle representation of each primitive has been prepared, the triangle normals are arranged to point out of the primitive volume. As all primitives are convex, it is straightforward to work out whether a point lies inside or outside of a primitive by calculating the dot-product between the normal of each triangle and a vector connecting the point in question with any point in the plane of the triangle. If and only if all of these dot-products are positive, the point lies inside the primitive volume. When determining whether a triangle lies inside or outside the CSG volume, special care has to be taken with coplanar triangles as the logic for rejecting or accepting such triangles differs depending on whether the triangle belongs to the main volume primitive, its parent or a child volume.

Finite machine floating-point precision also presents problems. Generally, whenever any comparison is made between two geometric objects (for example, testing the equality of two points or the co-planarity of two triangles) a tolerance must be included to allow for numerical round-off errors. Selecting the tolerance requires a tradeoff between acceptable error rates (most commonly, failure to remove an internal edge) and accidental removal of surfaces which are close but distinct. Using double-precision (8 byte) floating-point numbers, a tolerance of 10^{-6} is found to give good results.

To illustrate the operation of this algorithm, a CSG object comprising a box with 3 cylindrical holes is shown in figure 3. View (a) shows the primitives that comprise the volume along with their bounding boxes and constraint points. View (b) shows the primitives alone. Views (c) and (d) show the final CSG object, with wire frame and shaded rendering respectively. The CSG views are essential for appreciating the form of complex objects and checking that they have been correctly defined.

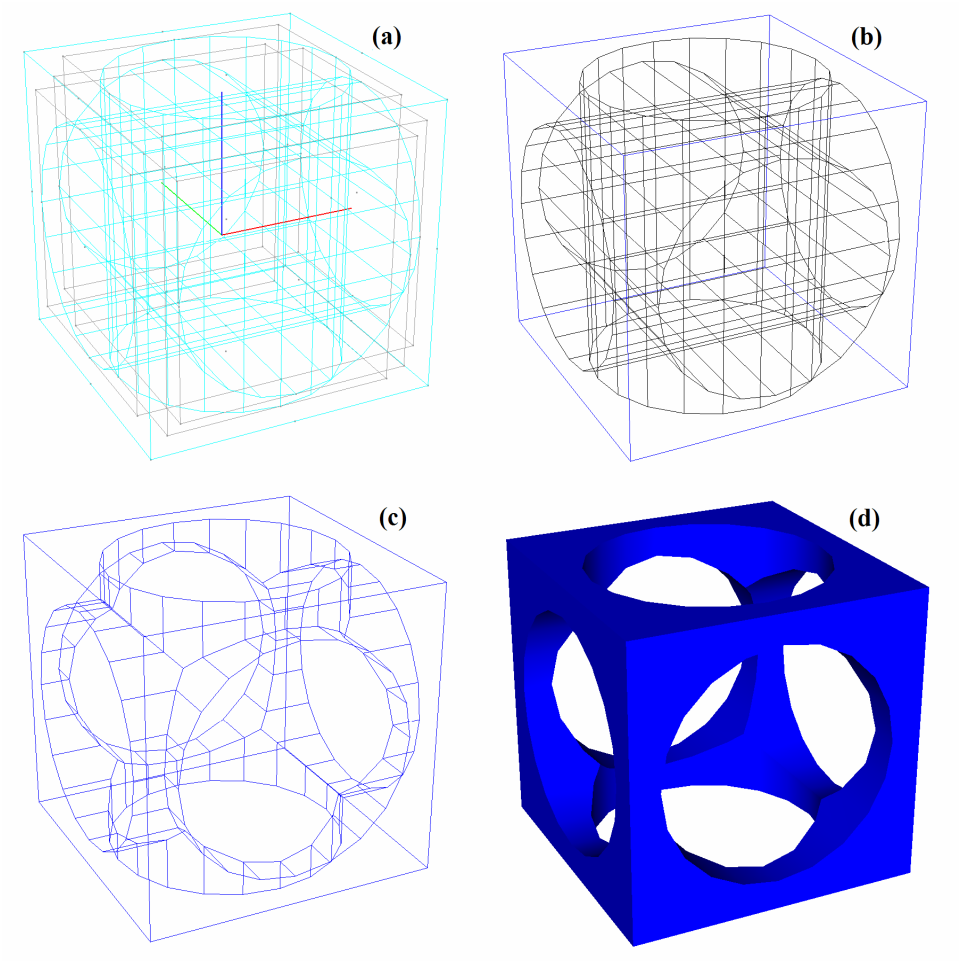

Figure 3. Four views of a complex CSG volume. (a) Primitives plus their bounding boxes (b) Primitives (c) Wire frame (d) Shaded solid.

4.2 Interactive event display

The user can run the EGSnrc and optical photon Monte Carlo codes interactively from within XPERT. Electron, positron and photon tracks are displayed, along with contributions to next-event, point-flux tallies. The energy and weight of any track segment can be checked by clicking on the track. Boolean combinations of up to 4 filters can be imposed, so that only events with particles reaching a certain volume, having a particular energy or contributing to a tally are displayed. Displayed tracks are overlaid on the problem geometry, which can be shown in wire frame, solid or combination views.

4.3 Tally result display

After a Monte Carlo result is completed, tally results can be overlaid onto the geometry display. This is particularly useful for spatially segmented tallies. Figure 4 illustrates the results from a sample problem. The model simulates a section of conveyor belt (blue) carrying coal (dark-grey, shown in wire frame). The belt is supported by steel rollers (pink). The elemental composition of the coal is measured by detecting gamma-rays that are induced by 14 MeV neutrons emitted by a sealed tube neutron generator situated below the belt. The gamma-ray detector is situated above the belt as shown.

A next-event, point-flux detector is used to estimate the gamma-ray flux spectrum at the detector position. The coloured points show individual flux contributions with the colour from blue to red indicating increasing weight. The grid of coloured squares at the front of the image shows the flux contributions integrated along the direction of travel of the belt, again coloured from blue to red. It can be seen that the strongest contributions come from the coal immediately beneath the detector. Contributions from scattering in the detector and neutron generator housings can also be seen.

The individual point-flux detector contributions are stored in the Monte Carlo output file. These are read in and used by the XPERT system to deduce the integrated contributions. This means that this information can be immediately recalculated and displayed without rerunning the main Monte Carlo simulation. Cuts can be placed on the energy and weight ranges of the point-flux contributions to be included, the orientation and position of the integration plane can be changed and the resolution of the integration grid can be altered. This provides a powerful tool for understanding the performance and optimizing the design of nuclear instruments.

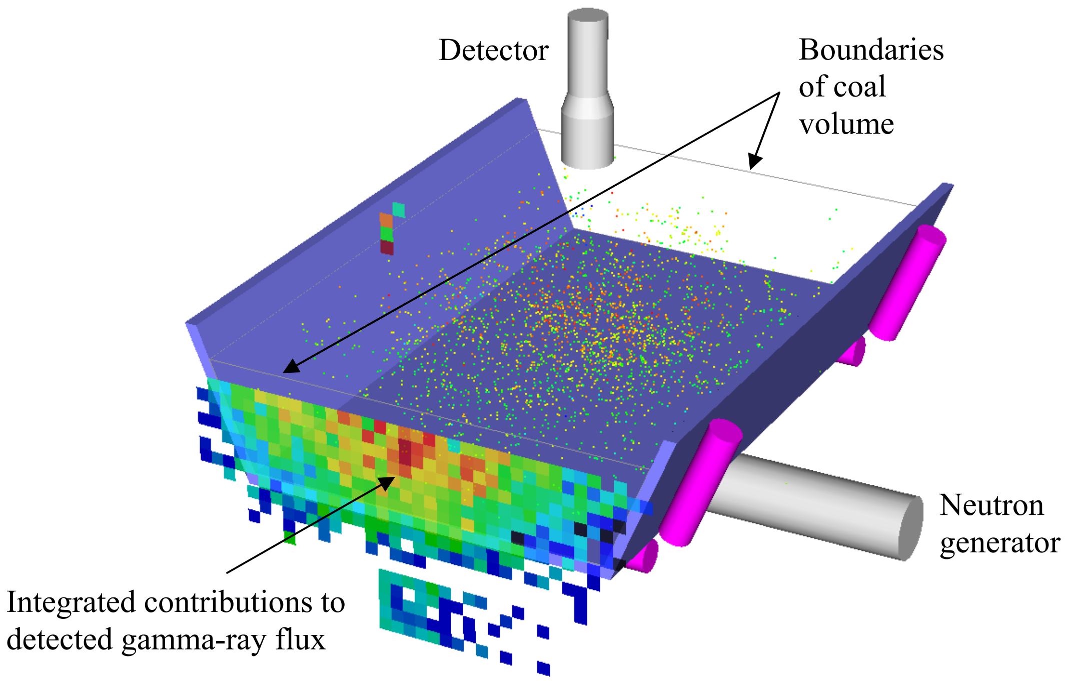
Detector
Boundaries of coal volume
Neutron generator
Integrated contributions to detected gamma-ray flux

Figure 4. Illustration of a neutron/gamma-ray coal analyzer, showing the origin of gamma-rays reaching the detector situated above the conveyor belt.

5 CONCLUSIONS

The user interface is an important part of a Monte Carlo particle transport code, particularly when problems with complex, 3-dimensional geometry models are being created. The XPERT code was developed to address three critical aspects. Firstly, a constraint-based CSG model allows for intuitive specification and positioning of basic volumes that comprise the design. The model automatically propagates changes made to one volume through the entire geometry as required and helps to ensure consistency of the geometry model. Secondly, an efficient, automated geometry checking tool helps find, identify and correct mistakes and ambiguities in the geometry model. Finally, conversion from the CSG model description to a polygonal BRep model allows the geometry to be interactively viewed and manipulated in real time, along with overlaid particle tracks and tally results.

6 ACKNOWLEDGEMENTS

The author would like to thank Greg Roach and Joel O’Dwyer for carrying out extensive testing of the XPERT system and their many useful comments, ideas and corrections.

7 REFERENCES

1. J. Knobloch, “ATLAS Computing,” *Comp. Phys. Comm.*, **110**, pp. 26-31 (1998).
2. I. Kawrakow and D. W. O. Rogers, “The EGSnrc Code System: Monte Carlo Simulation of Electron and Photon Transport,” *NRCC Report,* **PIRS-701** (2003).
3. J. Briesmeister, Ed., “MCNP^{TM} – A General Monte Carlo N-Particle Transport Code,” *Los Alamos National Laboratory Report,* **LA-12625-M** (1997).
4. “TCL Developed Exchange”, http://www.tcl.tk (2004).
5. E.R. Woodcock et al, “Techniques Used in the GEM Code for Monte Carlo Neutronics Calculations in Reactors and Other Systems,” *Proc. Conf. Applications of Computing Methods to Reactor Problems*, **ANL-7050** (1965).
6. L. Popescu, “A geometry modeling system for ray tracing or particle transport Monte Carlo simulation,” *Comp. Phys. Comm.*, **150(1)**, pp. 21-30 (2003)
7. E. Shuttleworth, “Fractal Geometry for Monte Carlo Tracking,” *Nucl. Ener.*, **31(2)**, pp 97-xxx
8. “OpenGL homepage”, http://www.opengl.org (2004).
9. J. Goldfeather et al, “Fast Constructive Solid Geometry in the Pixel-Powers Graphics System,” *Comp. Graphics (SIGGRAPH ’86 Proc.) ACM*, **20(4)**, pp. 107-116 (1986).
10. D. Epstein, F. Jansen and J. Rossignac, “Z-Buffer Rendering from CSG: The Trickle Algorithm,” *IBM Research Report,* **RC15182** (1989).
11. T. Wiegand, “Interactive Rendering of CSG Models,” *Comp. Graphics Forum,* **15(4)**, pp. 249-261 (1996).
12. N. Stewart, G. Leach and S. John, “A Z-Buffer CSG Rendering Algorithm for Convex Objects” *8th Int. Conf. In Central Europe on Comp. Graphics (WSCG 2000 Proc.)* (2000).