3.2.14

CombLayer : A fast parametric MCNP(X) model constructor

Stuart Ansell

Rutherford Appleton Labs, Chilton, Didcot OX11 0QX, U.K.

E-mail: stuart.ansell@stfc.ac.uk

Abstract.

MCNP(X) Monte Carlo neutronic modeling has now reached the level that large simulations of spallation sources starting from protons on target and recording neutrons at instrument detectors or outside biological shielding can simulated in one model. These models have the majority of the engineering aspects (e.g. pipework) described in detail. However, directly building an MCNP(X) input for a large geometry is highly time consuming and almost all the features of MCNP(X) that allow that process to be made simpler for the user (e.g. universes, lattices etc.) increase the simulation runtime by orders of magnitude.

CombLayer is a program designed to begin to overcome this problem. It ignores all the helper options in MCNP(X) and treats MCNP(X) as an assembler. Assembly-like geometry components can be rotated and repositioned, linked together, intersected and joined using a linkage system. Thus rapid production of complex MCNP(X) geometries that depend on a long list of variables and module flags, and the facilitation of tallies with appropriate variance reduction is possible.

CombLayer has built full models of facilities, with examples from ISIS TS1/TS2, SNS, ESS, Delft etc, typically with >2000 variables that can be changed within their range to carry out optimizations. Additionally, the object oriented form of all model components allows exchange of most parts to done with a simple command line flag. This facilitates "shopping list comparisons" e.g. a set of different cold moderators to be compared.

The code is publicly available at https://github.com/SAnsell/CombLayer.

1. Introduction

The Monte-Carlo MCNP(X) code [1] has been a mainstay of the neutronic community for many years and used in the design of both spallation sources such as the SNS and ISIS-TS2 as well as reactors e.g. ILL and Delft [2, 3, 4]. For the last 20 years there has been a steady increase in the realism and complexity of the models used, going from simple cubic or cylindrical moderators in a simple reflector to fully engineered moderators including pipework, pressure curvatures and corner rounds and detailed from the target to the instruments detectors. This has resulted in the number of objects and surfaces increasing from the 10s to the 100,000s. However, during the same time the mechanisms within MCNP(X) which allow the input of that geometry have not changed significantly. Surfaces are still constructed from simple quadratic infinite surfaces and the later addition of macrobodies have only added extremely primitive, non-extensible shaped surface groups e.g. the cuboid. Objects are constructed from a set of boolean operations on surfaces, the object must be completely defined and it not sufficient to allow previously defined objects to cut or overlap the new object.

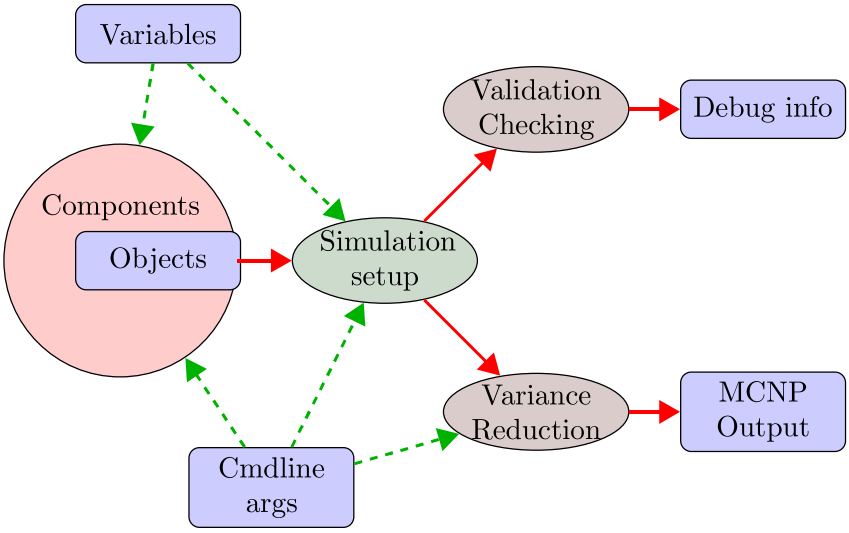

Figure 1. Diagram of the procedural flow and object construction within a CombLayer runtime execution. The green dashed arrows represent possible user input. The red solid arrows represent definitive directions of construction. First components and objects are built, simulation setup is carried out and then onwards to validity or variance reduction and output.

MCNP(X) suffers an additional problem when dealing with large assemblies, in that both the complementary operator and the universe system are not restricted. Consider three objects: A in B in C. If B is constructed using the complement of the outside of A (#A), and C is likewise constructed from B (#B), at runtime MCNP(X) needs to calculate all the surfaces in A, B and C to calculate points and lines in C. In large assemblies, this cascade dramatically reduces runtime performance or greatly increases the complexity of the modeling process.

The program CombLayer is being developed to alleviate some of these problems and in addition to be a basis program that can be extended/modified by others to be further improved.

1.1. CombLayer Architecture

The outline CombLayer architecture is shown in figure 1. The program requires users to effectively write their geometry into a C++ construction system. This is compiled into the program before running CombLayer. The output can then be influenced by a Turing complete variable system which can be set via XML files or via the command line

2. Object Construction

Volume objects within CombLayer are still constructed with a complete boolean volume description of each space in the same way that MCNP(X) treats each individual volume. Volumes are defined by combining *literals* (unique surfaces) with simple union and intersection (*operations*) to make a volume rule set. This is called a *function* within logic mathematics. All CombLayer volumes are initially pre-processed to remove non-unique surfaces and opposite plane surfaces against a global surface list. Then complementary components are expanded out.

CombLayer stores the object function as a binary tree structure. It is created by having the top node either an intersection or a union; this descends to other intersections or unions, before the leaf nodes are surface literals. The tree is partially sorted such that the minimum

number of changes of boolean type (intersection/union) occur between the head node and the surface literal, which implies that intersections and unions are gathered together. This allows the designation of two terms *surface level*, which indicates how many boolean operator changes that occur between the tree headnode and the point a surface is found, and *level object* which is the volume described by only those surface that share one particular *surface level*. The latter is commonly used for wrapping an object, finding its external surface etc.

CombLayer handles all of this internally, the only requirement from the user is to define the object either using the traditional MCNP(X) method, or as a composite object.

2.1. Object Organization

CombLayer allows the construction of objects within C++ classes that inherit from two base classes: *FixedComp* and *ContainedComp* or further derived versions of these two classes. The group of MCNP(X) objects built by this class is called a *component*.

Components are typically constructed in one class with this dual inheritance. The component is then automatically registered with the global (singleton) object register. The object register allows each component to be constructed as if it were an isolated simple MCNP(X) model. Surface numbers and objects numbers are automatically taken care, e.g. a cylinder component can be built with surfaces *1 2 -3* regardless of how many cylinder components or other components are required.

FixedComp provides an origin and an orthogonal basis set (X/Y/Z) that the geometry is being constructed with, which allows the object to be rotated and shifted in any direction, but constructed in a simple way. However, the real advantage of *FixedComp* is the provision of link points. A link point is a point on a surface which has an external axis associated with it. For example, a 4x4x4cm cube with the surface *pz 2* might have a link point on the surface at (0,0,2) with link direction (0,0,1). If the origin of the *FixedComp* is moved, or the object is rotated, these link points also follow the rotations/translations. This gives the advantage that when an new object is constructed, the new object is constructed relative to the direction and position of an existing link point. For example, a line of cubes can be created by constructing each (except the first) relative to an external link point. If the initial cube is rotated/shifted the whole line moves, but if one of the later cubes is shifted/rotated only the remaining cubes are reorientated. Further, since the link point provides a common surface interface, the cubes could be joined using this, so in the last case, the rotating cube becomes extended.

ContainedComp provides support for the external boundary of the a collection of MCNP(X) objects. It is an attempt to mitigate the cascade problem of complementary objects and universes within MCNP(X) whereby a sequence of inclusions between object A, B, C... will result in a MCNP(X) runtime degradation. The use of *ContainedComp* should give the same geometric result but not result in the runtime penalty.

When a component is built, an external boundary of surfaces is defined. The external boundary should wrap the volume of the component. In its simplest form, it can be constructed as a union of all the objects within the component, but commonly an obvious outer surface is available. For more complex object a *ContainedGroup* is available. This external boundary is not an object and does not appear within the MCNP(X) output model, but it can be used when a component could reside within another object, or if two components get so close that an intersection could take place.

ContainedComp is provided with a number of routines that make it more useful in building objects. First, if an intersection is certainly going to happen between two components (e.g. a moderator within a reflector), then the *ContainedComp* can be added directly to the containing component. Much more likely, is that the *ContainedComp* will be included in some (unknown number) of the objects that make up another component. This is carried out by using an appropriate attachSupport call, which automatically checks each of the objects within the

components to see if they overlap any part of the *ContainedComp*'s boundary. If that does happen then the full excluded boundary is added to the individual object. It is assumed that the object optimization will remove unnecessary surfaces later.

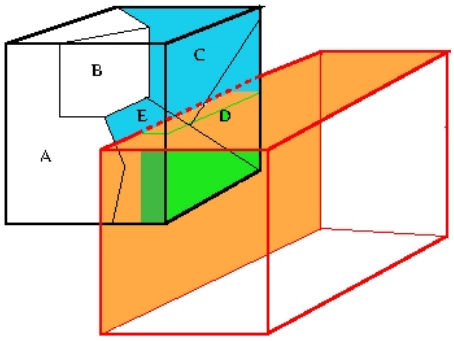

Figure 2. The intersection between an object components and a *ContainedComp*. Objects in blue (E/C/D) are the only object that get modified and they end up with only two of the surfaces (in orange) of the original ContainedComp

Figure 2 shows the consequence of the intersection of a *ContainedComp* with a different component after the object optimization process. It can be seen that *ContainedComp* are reductive, so the intersection of a ContainedComp with a component does not affect the objects own *ContainedComp*, so a sequence of component intersections are constructed, where a change to one component's composition or boundary never requires changes to another component.

The main drawback is runtime speed of building the models. In principle, it would be possible to intersect every component with every other component, but when developing models it is highly advantageous to keep the model building phase down to a few seconds. Thus a positive component-component test system is used, where the user has to state (in code) which objects are going to be tested together.

2.2. Object Optimization

During the MCNP(X) path tracking routines, the largest areas of CPU expenditure are: (i) calculating the side of a surface relative to a point, (ii) the intersection point of a line to a surface and (iii) exiting/entrance condition of a line on a surface intersection. All three of these can be dramatically reduced by reducing the number of *literals* (unique surfaces) within a volume description. If this can be achieved, both the overhead in calculating the object and number of other cells that need to be investigated when calculating the exit of a track are reduced. It is this latter overhead that can increase very dramatically as the complexity of the model is increased, with particular note to pipework and other connectible items.

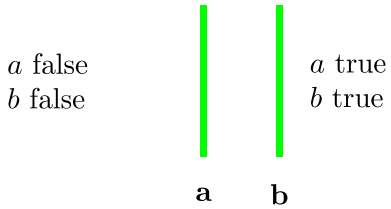

Figure 3. The planes a and b are parallel so there is an impled relation ship between them. The sense of the planes are arbitarily decided to be true towards the right.

CombLayer provides two operations for improving the number of literals; both need the pre-operational step of expanding the object function based on absolute implications. In this process, all pairs of literals (a,b) which imply the other are expanded, e.g. the two parallel planes shown in figure 3 have the property that if plane b is true, then plane a is also true, and likewise if a is false b is also false. This is expressed in standard boolean algebraic form as (using + for

union, $'$ for negation).

$$\begin{aligned} b &\implies a \equiv b' + a \\ a' &\implies b' \equiv b + a' \end{aligned} \tag{1}$$

These rules in the union form are then intersected with the existing volume object resulting in a obviously longer expression, but surprisingly because MNCP(X) calculates only those surfaces that are required for a particular condition to be determined, this expanded form results in a slightly faster executable model in MCNP(X).

This object can be further improved by removing unnecessary literals. This is done by determining the Shannon expansion for all repeated literals (regardless of +/-ve sense). The Shannon expansion of the literal a gives the logical function in the form: $a(A) + a'(B)$, where A and B are logical functions without the literal a. In the case, that $A \equiv B$, literal a can be removed from the function.

Further optimization could be carried out if 2-factor factorization and weak division of factors were available and it is expected that this capability will be added soon.

2.3. Model Variables

An essential part of geometric modeling is having control over all the parameters that make up the model which allows the user to change any parameter without the model becoming broken. CombLayer tries to ensure this via two mechanisms. The first is that there is a global database of variables which all have a type and a default value. Variables can be floating point number, integers, strings or 3D vectors. Those variables can be expressed either as values or as mathematical functions of other variables or programming constructs. Second, combined with the *FixedComp* link points and surfaces, most objects are extremely robust to change of position and size. If an object is likely to be eliminated then additional code has to be added to components to deal with this issue, but typical size change can be accommodated by standard overlap tests and exclusion on all nearby objects.

2.4. Pipework

Adding pipework to a model is such a common requirement that a specific module has been written to assist in this operation. In CombLayer a pipe is considered as a multi-layered object that follows a path through the model but has a constant shape form (e.g. cylindrical/square). The shape can change size (e.g. a pipe going from a small radius to a large radius) but the shape is considered unchanged. Additionally, the use of simple flags can turn various layers on and off at each junction.

The pipework is added by starting a pipe relative to a link point or central origin of a component. Obviously its true starting point can be offset from this link point. Then further points are defined either by another link point, or by 3D-Vector offsets until the pipe's termination point. The pipe is bent at each turn by introducing a dividing plane, even if that plane produces zero deflection. If the pipe goes through a *LayerComp* object it may be joined to each layer in turn [figure 4].

3. Post-Geometry Construction

Once the geometry has been constructed, further processing in necessary to add source terms, tallies and carry out variance reduction. CombLayer supports most tallies and some source terms, as currently written. Since CombLayer renumbers surfaces, objects and most object's final positions which are dependent on the state of many variables, it takes advantage of the *FixedComp* link points, link surfaces and the objectRegister for setting tallies. Additionally, if

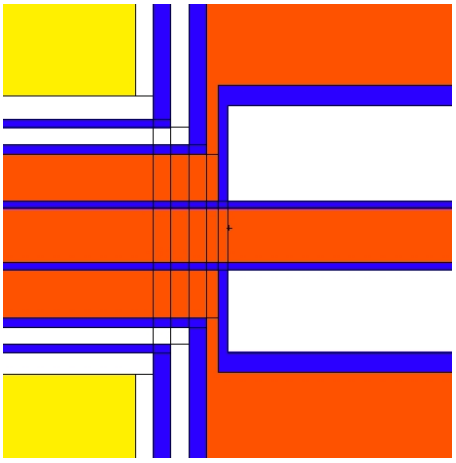

Figure 4. The pipe join with a multi-layered moderator. The requirement is that the moderator vessel is a *LayerComp* and the pipe join flag is set for each of the pipe/layer intersects. The dividing planes of each of the pipe segments can be seen within the join region.

tallies are not set via this mechanism, significant extra effort is required to set up an appropriate variance reduction mechanism.

The simplest tallies to set up are point tallies. These can be constructed as an offset 3D-vector distance from any link-point. In addition, they support area-point tallies and windowed point tallies which are common MCNP(X) modifications. Similarly surface tallies can be set up if a link-surface is required, although more difficult if not, and cell (flux/heat) tallies can be added via the objectRegister. Finally, XML tallies can be added for any other tally type or addition.

3.1. Variance Reduction

There is no complete variance reduction method in CombLayer, rather a number of enablers to construct a variance reduction system to suit the type of problem being studied. However, significant effort has been made to enable many appropriate questions to be asked of a geometric system.

Firstly, it is possible to determine an approximate *centre* of most objects and components. This is done by iterating over all the surfaces in an object to determine the triple surface intersections. Three non-parallel planes will intersect at a point; a cylinder and two planes typically intersect at two points etc. These points are then summed to a centre of mass if the point is on a true surface of the object.

The centre object points can be used in a number of ways to help with variance reductions. The first is via the cell based weight window system. For each cell it is possible to determine a biased probability to transport to any other cell based on the distance between the centres and the material in the path between the centres. Following this direct calculation of all object pairs, a Markov chain method can be used to track the cell to tally position bias for the weight window value. Different energy values can have different tracking cross sections associated with them which in turn modify the weight window result. Obviously, this is not a replacement for full adjoint variance reduction method [5], however it is very fast to use and requires minimal setup effort.

4. Conclusions

CombLayer is an extensible object-orientated system for building MCNP(X) models. Its main strength is to allow the rapid construction of complex non-repeating structures such as found in spallation sources and reactor assemblies. These structures can be linked in simple ways that allow the size and position of almost any object to change without breaking the model or having to do additional work to make the model work.

It has been used to build full assembly models from proton targets to instrument layouts in one model which could be rapidly run in a single MCNP(X) model. These models have had upto 100,000 MCNP(X) objects within them but could still be run effectively.

The code released under the GNU public license GPL3 and is available at https://github.com/SAnsell/CombLayer.

References

[1] X-5 Monte Calro team 2003 MCNP - A General Monte Carlo N-Particle Transport Code, version 5 Tech. rep. Los Alamos

[2] Gallmeier F X, Ferguson P D, Iverson E B, Popova I I and Lu W 2006 *Nuclear Instruments and Methods in Physics A* **562** 946–949

[3] Lu W, Ferguson P D, Iverson E B, Gallmeier F X and Popova I I 2008 *Journal of Nuclear Materials* **377** 268–274

[4] Ansell S 2007 *Proceedings of ICANS-XVII* pp 660–668

[5] Wagner* J C and Haghighat A 1998 *Nuclear Science and Engineering* **128** 186–208