CombLayer Guide

Stuart Ansell

November 30, 2017

Contents

1 Introduction

CombLayer is designed to facilitate the rapid production of complex MCNP(X) models that depend on a long list of ranged variables and a number of module flags. It is also intended to help with placement of tallies, maintaining consistant material files and some variance reduction.

1.1 Coding Convensions

CombLayer has some coding convensions beyond the standard Scott Myers Efficient C++ convensions [**?**]. These are typcially there for two reasons (i) that in a model-build system, a rapid build time is essential since it is nearly impossible to have a sub-test framework for any component as the whole MCNP(X) model is required to check if is it valid, (ii) the code is intended to be used without complete understanding. Therefore as much as possible, each component is independent without code repetition. Back-references are to be minimized both in the run-time calling path and in the code build dependencies.

1.1.1 Include files

Include files (.h) are forbidden to include other files. This does several things (a) it reduces the *dependency hell* where it is almost impossible to find the definition of a function and what it depends on. (b) optimization of the include tree can be carried out and dependency continuously observered.

Namespaces are a good method of removing global name pollution but many other C++ programs allows *using namespace X*, this is almost 100% forbidden except in the test for that particular namespace unit. This also applies to boost, stl, tr1 etc, to which helps distinguish external functions and domains.

2 Layout

The basic program structure is given by figure 1. The main program structure is normally copied from an existing project and the areas are constructed by the user. It is normal flow is to to call functions that: (i) define new input options to enter parameters from the command line, (ii) variables that your project is going to use, (iii) build the geometry via a call to a makeProject functoin (iv) set up tallies (v) generate variance reduction.There are other ways to construct the system but this allows a degree of autonomy from tallies/variance reduction and producing an appropiate output.

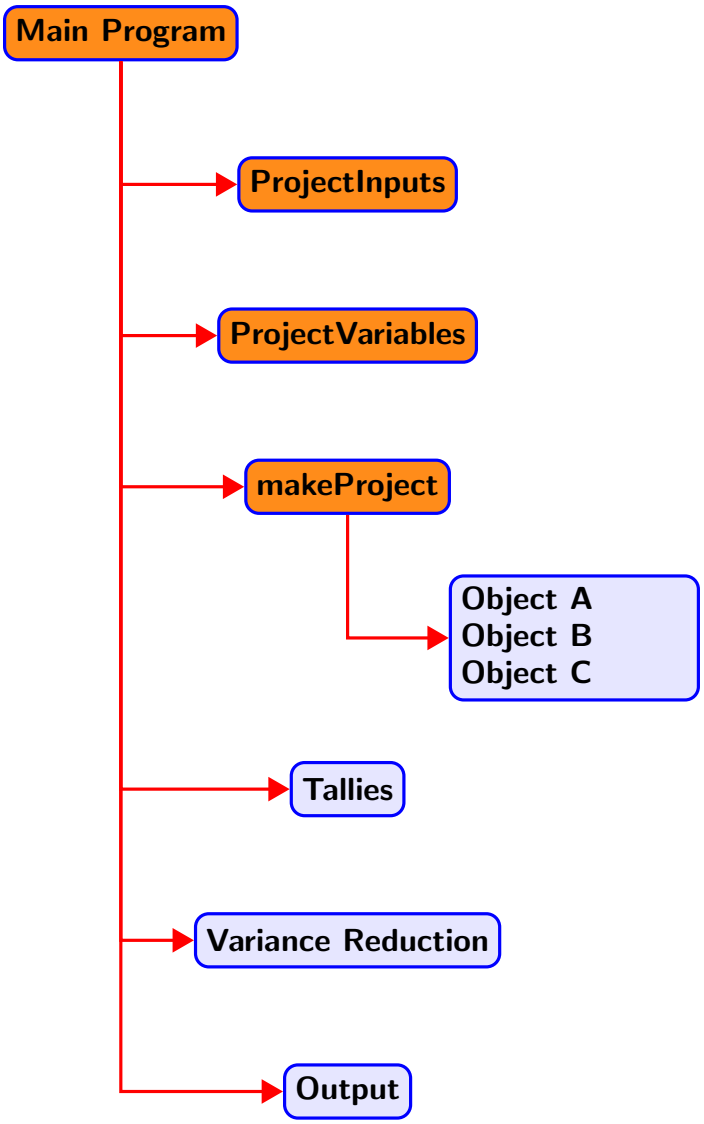


Figure 1: The main program calling sequence is shown. The parts in orange, are expected to be constructed by the user. Bespoke objects can be added for a project but it is not necessary.

2.1 Main

The main function for CombLayer follows a relatively linear template. Consider the example

```
int
main(int argc,char* argv[])
{
  int exitFlag(0);                    // Value on exit
  ELog::RegMethod RControl("","main");
  mainSystem::activateLogging(RControl);
  std::string Oname;
  std::vector<std::string> Names;
  std::map<std::string,std::string> Values;

  // PROCESS INPUT:
  InputControl::mainVector(argc,argv,Names);
  mainSystem::inputParam IParam;
  createPipeInputs(IParam);

  Simulation* SimPtr=createSimulation(IParam,Names,Oname);
  if (!SimPtr) return -1;

  // The big variable setting
  setVariable::PipeVariables(SimPtr->getDataBase());
  InputModifications(SimPtr,IParam,Names);

  // Definitions section
  int MCIndex(0);
  const int multi=IParam.getValue<int>("multi");
  try
    {
      SimPtr->resetAll();

      pipeSystem::makePipe pipeObj;

      World::createOuterObjects(*SimPtr);
      pipeObj.build(SimPtr,IParam);
      SDef::sourceSelection(*SimPtr,IParam);

      SimPtr->removeComplements();
      SimPtr->removeDeadSurfaces(0);
      ModelSupport::setDefaultPhysics(*SimPtr,IParam);

      const int renumCellWork=tallySelection(*SimPtr,IParam);
      SimPtr->masterRotation();
      if (createVTK(IParam,SimPtr,Oname))
        {
          delete SimPtr;
          ModelSupport::objectRegister::Instance().reset();
          ModelSupport::surfIndex::Instance().reset();
          return 0;
        }

      if (IParam.flag("endf"))
        SimPtr->setENDF7();

      SimProcess::importanceSim(*SimPtr,IParam);
      SimProcess::inputPatternSim(*SimPtr,IParam); // energy cut etc

      if (renumCellWork)
        tallyRenumberWork(*SimPtr,IParam);
      tallyModification(*SimPtr,IParam);

```

```
61        if (IParam.flag("cinder"))
62          SimPtr->setForCinder();
63
64        // Ensure we done loop
65        do
66          {
67            SimProcess::writeIndexSim(*SimPtr,Oname,MCIndex);
68            MCIndex++;
69          }
70        while(MCIndex<multi);
71
72        exitFlag=SimProcess::processExitChecks(*SimPtr,IParam);
73        ModelSupport::calcVolumes(SimPtr,IParam);
74        ModelSupport::objectRegister::Instance().write("ObjectRegister.txt");
75      }
76    catch (ColErr::ExitAbort& EA)
77      {
78        if (!EA.pathFlag())
79          ELog::EM<<"Exiting from "<<EA.what()<<ELog::endCrit;
80        exitFlag=-2;
81      }
82    catch (ColErr::ExBase& A)
83      {
84        ELog::EM<<"EXCEPTION FAILURE :: "
85                <<A.what()<<ELog::endCrit;
86        exitFlag= -1;
87      }
88    delete SimPtr;
89    ModelSupport::objectRegister::Instance().reset();
90    ModelSupport::surfIndex::Instance().reset();
91
92    return exitFlag;
93  }
94  \label{MainProg}
```

The Main program given in listing **??** highlights the areas that the user should be creating. The remainder of the main() function deals with trapping exceptions, loggin and building variance reduction and tallies into the model.

(i) **createPipeInputs** is a function to define which command line options [above the standard ones] this model should support. It doesn't do anything with them, just a list of options, number of arguements they can take and any default values that the options should take. All options defined here are access from the command line option with a - sign. E.g. *-r* as a renumber operation. In this form of the program, if the main program is run without any options, a list and very brief description of each option is shown (e.g. execute *./pipe*).

If no additional options are required, a call to *createInputs(IParam)* would be expected. Significant restructuring would need to take place to avoid that call.

(ii) setVariable::PipeVariables is the method that registers and sets a default value for all the variables that the model will use.

(iii) makePipe pipeObj and pipeObj.buid(SimPtr,IParam) are the main geometry building calls. Typically 100% of the geometry is built in this zone. It is not a place for tallies, variance reduction and other non-geometry items.

3 Installation

CombLayer is predominately written for the Linux platform using C++ compilers that support C++11 or greater. The code is available from
https://github.com/SAnsell/CombLayer,
either as a download of a zip file or by cloning/pulling the git repository.

3.1 Requirments

CombLayer needs to have the GNU Scietific Library [GSL] and the boost::regex system along with the STL libraries from your C++ compiler. The GSL can be avoided with the -NS flag in the getMk.pl and the CMake.pl script but some functionality will be lost, particularly in the choice of variance reduction methods.

Additionally, the primary build system uses cmake. There is another that just uses make but is significantly more time-consuming.

Functional documentation is supported using Doxygen and the construction of new cmake text files can be done via PERL scripts.

Currenly it is know that gcc version 4.6 and above can compile CombLayer as can clang (all tested versions). gcc 4.4 which is often the default on RedHat systems (2015) does not work.

3.2 Basic build method

If a clean directory is made and then the .zip file is uncompressed, the following commands should build a version of CombLayer.

```
./CMake.pl
cmake ./
make
```

This should make a number of executables, e.g. ess, simple, fullBuild etc. These can be used to make a simple model with commands like

```
./simple -r AA
```

This will produce an output file AA1.x which is a MCNP model.

4 Link system

CombLayers geometry is composed of a set of objects that have slightly stronger rules than a typical MCNPX model. Obviously any MCNPX model can be represented as a CombLayer model and in the extreme case that is done by defining one object to contain the MCNPX model. However, the little benefit would be derived from such an approach.

The basic geometric system is to build a number of geometric classes and construct the model by incorporating those into the desired configuration. Each geometric class is designed to be built and an arbitary position and rotation, be of an undetermined number, and interact with its surroundings in a well defined manor.

In object orientated programming, functional rules and properties are normally added to objects by inherritance. CombLayer follows that pattern. As such most geometry item classes inherit from base classes within the attachSystem namespace.

4.1 AttachSystem Namespace

The CombLayer system is built around the interaction of FixedComp units, ContainedComp units and LinkUnits. The use of these and their interactions are the basic geometric building tools. These object reside within the attachSystem namespace.

Almost any geometric item can be designated as a FixedComp object. This is done by public inherriting from directly from the FixedComp, or by inherriting from one of the more specialised attachSystem objects e.g. TwinComp or LinearComp.

4.2 FixedComp

The basic FixedComp object holds the origin and the orthoganal basis set (X/Y/Z) for the geometry item being built. In addition it holds a number of LinkUnits which provide information about the outer (and/or inner) surfaces and positions on the geometric item.

As with all Object-Orientated (OO) constructions their is an implicit contract that the inherited object should adhere to. This is normally expressed as the *Liskov Substitution principle*: This principle states that functions that use pointers/references to the base object must be able to use the objects of derived classes without knowing it. In this case, that means that modification of the origin or the basis set should not invalidate it and that the object should do the expected thing. E.g. if the origin is shifted by 10 cm in the X direction the object should move by 10cm in the X direction. It also means that the basis set must remain orthogonal at all time.

Other than providing an origin and an basis set, the FixedComp has a number of link points. The link points are there to define joining surfaces, points and directions. Each link point defines all three parts.

For example a cube might have 6 linkUnits, and each linkUnit would have a point at the centre of a face, a direction that is normal to the face pointing outwards and a surface definition that is the surface pointing outwards. [Note that in the case that the link points define an inner volume, for example in a vacuum vessel, then the surfaces/normals should point towards the centre.]

The actual link surface does not need to be a simple surface. In the case, that an external surface needs multiple surfaces to define the external contact these can be entered into a link-rule. For example, if the cube above was replaces with a box with two cylindrical surfaces the link surface would be defined as the out going cylinder intersection with a plane choosing the side.

In the case of an equiry for the linkSurface (e.g. to do an line intersection) then it is the first surface that takes presidence. However, all actions can be carried out on the link-rule including line intersections etc.

4.3 ContainedComp

The ContainedComp defined both the external and interal enclosed volume of the geometric item. It is most often used to exclude the item from a larger enclosing geometric object: e.g. A moderator will be excluded from a reflector, or it can be used to exclude a part of the geometric item from another geometric object. E.g. two pipes which overlap can have one exclude itself from the other.

In CombLayer, the ContainedComp are considered the primary geometric item, i.e. it is the ContainedComp that is removed from the other items. However, it is used in a two stage process wereby cells are registered to be updated by the ContainedComp at a later date. This was to allow forward dependency planning but has more or less been superseded by the attachControl system.

5 Model Runtime control

C++ programs start from the main() function and in CombLayer the runtime control has been keep mostly in the main() function. Clearly that could be further refactored out but CombLayer lacks the sophisticated top level type abstraction that is required to do this in a generic way, so copy/pasted structure is used with variance to the particular model required. The sole advantage of the abscence of a top level abstraction is that the user is the freedom in writing new objects which allows other programs to be incorporated by making their main function a minor function and directly calling.

The structure of two example main()s will be compared from the units that exist with the stanard CombLayer distribution. That is *bilbau.cxx* and *reactor.cxx*. These build the delft reactor model and the Biblau low energy spallation source.

First part of the code is along list of #include's. They are the mainly dependency list of the objects *Simulation, weightManager, and tallySelector.* This can and should be copied at will. Do not make an file with them all in [see 1.1.1].

At the end of the include section there is typically, one or two model specific includes. These normally include *makeXXX.h* file and anything that they directly depend on. In the case of bilbau it is just *makeBib.h* whilst for reactor it is both *makeDelft.h* and *ReactorGrid.h*.

5.1 makeModel

The makeModel object is the place that creates, initializes and manages inquires for the instances of all the geometric components. Primary objects need to be created and registered with the objectRegister **??**. The makeModel component is

Tallies are the fundamental reason for running MCNPX. However, the manor in which MCNPX specifies tallies is not compatible with a vairiable defined model because in most cases the required tally is relative to an object whose position is unknown.

This problem has been addressed by allowing most tallies to use the FixedComp link system.

5.2 Tally System

The tally system is accessed either by a simple command line menu system, or via an XML file. The command line help system is very primative but can remind the user of the bais

5.3 Point Tally

Point tallies are fundamentally a 3D vector in space. In CombLayer, there are three levels of position available: (a) Real MCNP(X) output position, (b) CombLayer master origin position before master rotation, and (c) relative position to an object. Both (a) and (c) are well supported, however, to do option (b) there needs to be some real care with the layout of the calling sequence in the main() function. The fullBuild.cxx example is a suitable option to follow, but checking will be needed.

5.3.1 Free Point tally

The simplest way to put a point into CombLayer is to use a free point.

```
./prog -T point free 'Vec3D(300.0,10.0,5.0)' Output
```

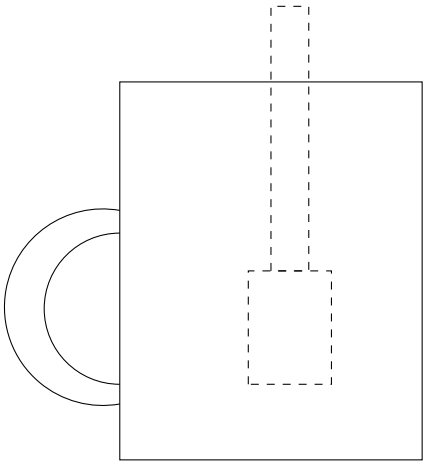


Figure 2: addToInsertForced: all cells of Spoon are excluded from all cells of Mug including non intersecting cells. In this example, the Spoon is excluded from the Mug handle even though they do not intersect.

This creates a point tally at (300,10,5) in the final output using neutron tallies with the default energy and time binning system.

- Get real surface number by its relative number: SMap.realSurf(divIndex+103) (see createLinks methods)

5.4 How to put one object into another

Suppose, we are inserting Spoon into Mug. Mug is made up of N cells. Spoon is made of one contained component with outer surface. CombLayer provides several methods to put one object into another:

```
attachSystem::addToInsertForced(System,    *Mug, *Spoon);
attachSystem::addToInsertSurfCtrl(System, *Mug, *Spoon);
attachSystem::addToInsertControl(System,   *Mug, *Spoon);
attachSystem::addToInsertLineCtrl(System, *Mug, *Spoon);
```

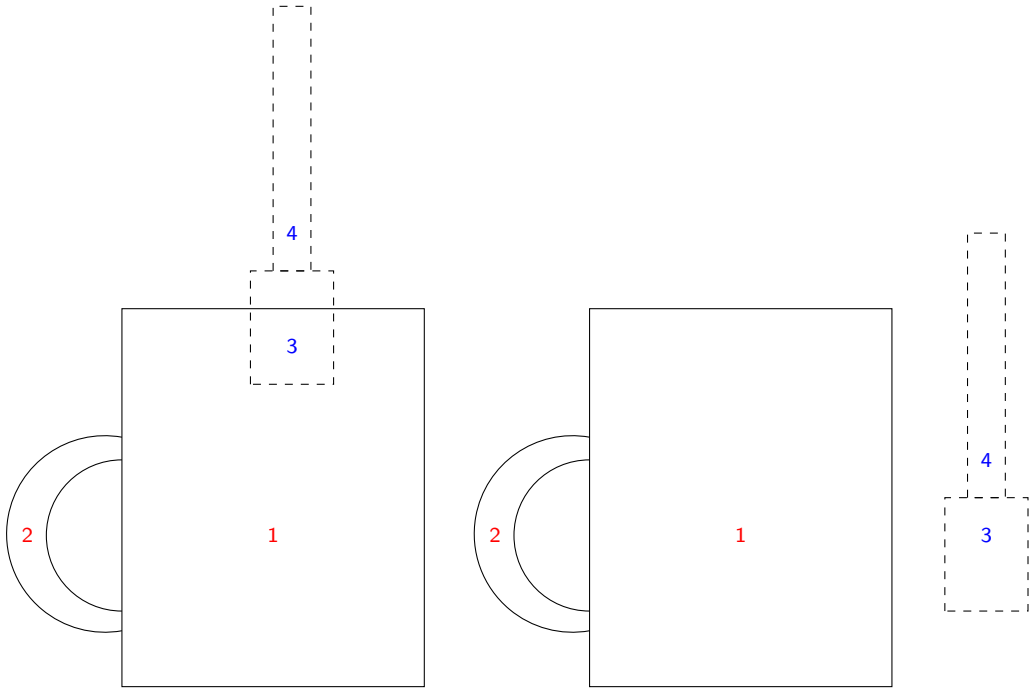
5.4.1 addToInsertForced

The outer surface of the Spoon is excluded from the HeadRule of every single cell of Mug. Even if Mug contains cells which do not intersect with Spoon (e.g. its handle, see Fig. 2). *Forced* means *do it and do not think about it*, but at the same time it means that *I have got something wrong somewhere.* Normally this is that insufficient link points have been added to the object, or that the object is a set of split (single cell) volumes. However, there is the additional problem that the model may not be correctly constructed at this point, so that the other options seem not to work. This can be checked by adding a *SimProcess::writeIndexSim(System,"OutputFilename.txt",0);* in the code just before the call to insertForced. If there are undefined volumes then the model is not in a state that any of the *addToInsert* algorithms except *addToInsertForced* can be used.

5.4.2 addToInsertSurfCtrl

The objective of this function is to use the surface intersections between Mug and spoon to determine which cells within Mug intersect the ContainedComp of spoon. The process is done on a cell - ContainedComp level.

The process is as follows:



(a) Cell 3 is excluded from cell 1; cells 2 and 4 not changed

(b) None of the cells are modified

Figure 3: addToInsertSurfCtrl: only needed cells are excluded

1. Deconvolves both the Spoon's containedComponent boundary into surfaces.
2. Deconvolves the Mug into surfaces.
3. Triple Loops over surfaces of both Mug and Spoon
   (a) Calculate intersection of each surface:surface:surface triplet
   (b) If a point is within CC and the Mug Cells exclude the CC from the MCell

Thus the spoon is inserted only into those cells of Mug which it intersects (Fig. 3).

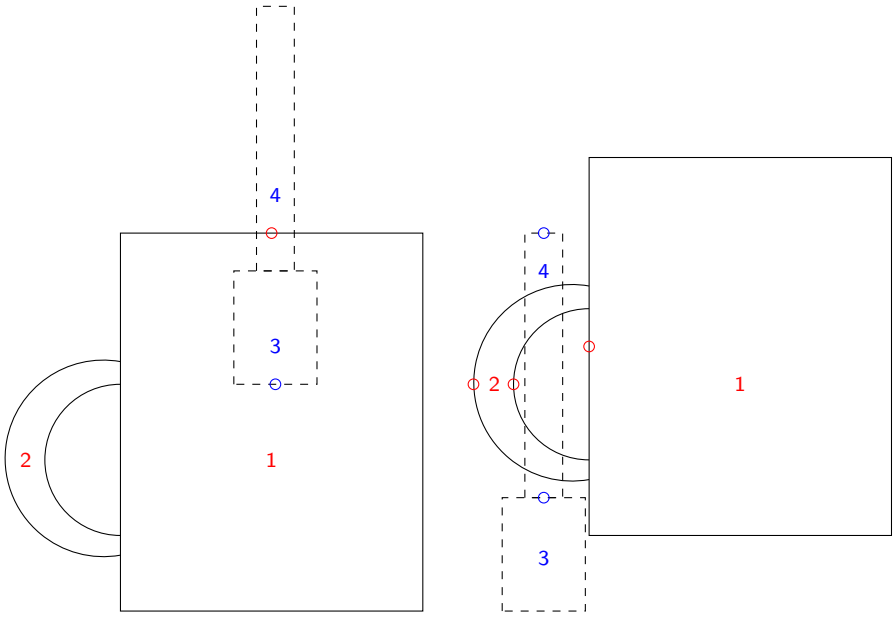
It is not always better to call `addToInsertSurfCtrl` instead of `addToInsertForced` in cases that if is certain that an intersection can must take place (particularly if the CC / Inserting cells have large numbers of surfaces).

`addToInsertSurfCtrl` is a very expensive function to call, because you have to check all the surface triplets. So, it runs significantly slower than addToInsertForced, but the geometry will be faster.

The two remaining methods provide similar functionality but with less computational overhead, however, there are cell constructs which will cause them to fail.

5.4.3 addToInsertControl

It's a very simple method. The link points from Spoon are uses as a test for each of the cells within Mug. The method checks if any of these link point fit inside each of the cells of Mug. If it does, then it cuts Spoon from the Mug cell (Fig. 4). It is possible to add a vector of link points to check as a parameter to limit the search.



(a) Cell 3 is excluded from cell 1 since cell 3 has a link point inside cell 1. Cell 4 is excluded from cell 1 since Mug has a link point which is located inside cell 4. Cells 2 is not modifed.

(b) Intersection is not made because none of the link points are located in another component. Geometry is broken. In order to fix it, either define more link points or use another volume inclusion method.

Figure 4: addToInsertControl: intersection relies on the link points

5.4.4 addToInsertLineCtrl

If we have a (big) contained component (Mug) and some (small) object which clips it (Spoon) and we are not certain that the link points are contained within each Mug cell, then **addToInsertControl** can not be reliably used. However, addToInsertLineCtrl adds an additional check to addToInsertControl, because it constructs all the connecting lines between the link points, and if the line intersects the other cell, an intersection is made (Fig. 5).

6 Components

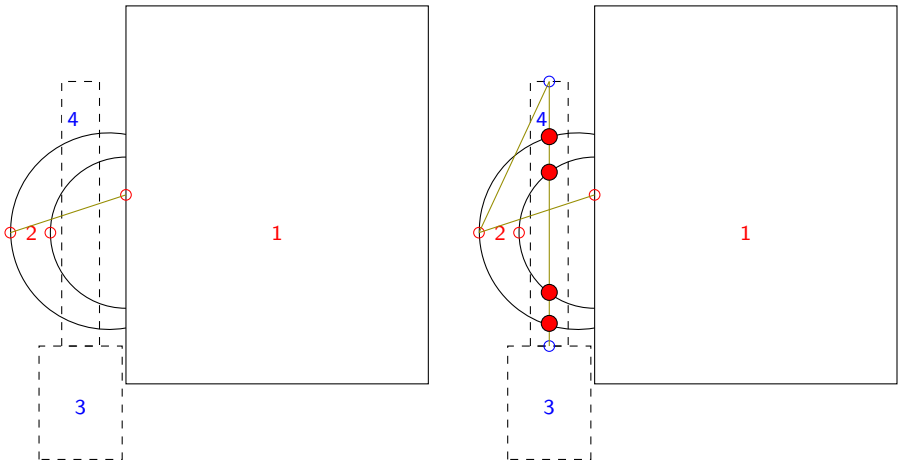
6.1 ObjectRegister

The objectRegister is a singleton object [it should be per simulation], which keeps each and then deletes when at its lifetime end, each object registered with it. It only accepts two types of object, a dummy name object and a FixedComp object.

If a dummy object is required, the name (and possibly number) of the object is provided and the objectRegister singleton provides a unique range of cell and surface numbers, typically 10,000 units of each, but can be user selected. This is its only responsibility and to ensure that the name is unique.

Significantly more complex is the FixedComp registration, in this case a *std::shared_ptr* of FixedComp must be provided by the calling method. Obviously, for a shared_ptr the object memory must be allocated, i.e. an initial `new object(...)` is normally called directly or previously. A typical structrue might be:

```
std::shared_ptr<BeamPipe> A  = new BeamPipe("LongPipe");
```



(a) Intersection is not made because the line connecting two link points intersects cell 4 outside of a cell belonging to the Mug

(b) Cell 4 is excluded from cell 2 since at least one line connecting two link points intersects the Mug handle at a point which is valid within the spoon (shown by red filled circles)

Figure 5: addToInsertLineCtrl: intersection relies on the lines connecting link points (not all link points and connections are shown).

```
ModelSupport::objectRegister& OR=
    ModelSupport::objectRegister::Instance();

OR.addObject(A);
```

From this example, the BeamPipe class is inherrited from FixedComp, this is manditory. A tempory reference *OR* is created by calling the static Instance() method. All singletons in CombLayer provide an Instance() method for this purpose. Then the object pointer is referenced to the objectRegiste with *addObject*.

However, hidden from view is a call to objectRegister in FixedComp's constructor, which is certain to be called as all registered object must derive from this class. That occures during the operator new call and results in the allocation of the cell/surface numerical range. If it is necessary to trap that error, the try/catch block must be around the new opertor. The main exception that is possible if an existing object already exists with the same name.

6.2 EXT Command

MCNP(X) provides the EXT card for biasing the direction of the particles after collisions. The card can be configured with a stretching parameter value between -1.0 and 1.0 and an optional vector or direction associated with it. If a vector is not given the stretching parameter is applied in the direction of the neutron travel.

MCNP(X) only accepts the direction to be X,Y,Z which is highly limiting in the CombLayer environment, so it is only partially supported.

The other two options vector and non-vector are supported.

6.3 -wExt entry

The first method of entry is via the command line option -wExt. This command takes a sequence of additional values which are split into a *zone* and *type* region. The *zone* region is based on the cells that are to be biased.

This can be give with the commands:

6.3.1 Zone

- **all** : Apply to all non-void cells
- **Object [name]** : Apply to all objects within the object name
- **Cell [Range]** : Apply to all objects within the range

Name can be a compound name of type objectName:CellMapName. This would just select those cells within the cellMap unit of the particular object.

6.3.2 Zone

7 User guide

This section describes how to use CombLayer from a user's (i.e. non-developer) point of view. In this guide, it is assumed that the user has no C++ or coding experience.

The guide is focused on the ESS model, which can be generated by running.

```
./ess -r modelOut
```

This command produces the `MCNP` input file `modelOut1.x` as well as two other files: `ObjectRegister.txt` and `Renumber.txt`.

The single flag `-r` is optional for MCNP6 as it causes the objects and surfaces in MCNP to be renumbered sequentially and to fit within the 100,000 object/surface limits of MCNPX.

7.1 Variables

In the beginning of the input file there is a commented list of variables which define the geometry:

```
c ------------------------------------------------
c ----------------- VARIABLE CARDS ---------------
c ------------------------------------------------
c ABunkerFloorDepth 120
c ABunkerFloorThick 100
c ABunkerLeftAngle 0
c ABunkerLeftPhase -65
c ABunkerNLayers 1
...
```

The variable name consists of the component name and its corresponding parameter. For instance, the first variable `ABunkerFloorDepth` in the list above sets the floor depth of the component called `ABunker`.

Only variables that have needed to be examined are included in this output. Several components are optional and if not built then their corresponding variables are not seen in the output. All variables start with an initial value and stateless variables are prohibited. Most variables are defined appropiately within the CombLayer program and obviously can be changed by editing the code and recompiling. However for a simple variable change that is excessive work so there are a number of methods to change variables from the command line.

7.1.1 How to change variables

Any of these variables can be changed either via a command line arguments or an XML file.

As an example, consider changing the Beryllium reflector height. First of all, we need to find out which variable we need to change and therefore find out the name of the Be reflector component in CombLayer.

To do this, open the `MCNP` geometry and click on any Be reflector cell. Currently, it's cell number 5 (exact number depends on the CombLayer version you are using).

Now we need to find out which component this cell belongs to. Find this cell number in the `Renumber.txt` file:

```
grep " 5 " Renumber.txt
Surf Change:1000006 5
Cell Changed :1000005 5 Object:BeRef (topBe)
```

It shows that the corresponding Be reflector object is called `BeRef`.

Now we need to find out which `BeRef` variable is responsible for its height:

```
grep BeRef a1.x
c BeRefHeight 74.2
c BeRefLowRefMat Be5H2O
c BeRefLowWallMat Stainless304
c BeRefRadius 34.3
c BeRefTargSepMat Void
c BeRefTopRefMat Be5H2O
c BeRefTopWallMat Stainless304
c BeRefWallThick 3
c BeRefWallThickLow 0
```

We can guess from this list that the variable we need is called `BeRefHeight`.

Changing variables via command line In order to change a variable via command line arguments, run:

```
  ./ess -r -v BeRefHeight 50 modelOut
```

Several variables can be changed, e.g.:

```
  ./ess -r -v BeRefHeight 50 -v BeRefRadius 35 modelOut
```

Note that it is possible that you miss spell one of the variables, if this is the case then you will be presented

```
./ess -r -v BeRefHHH 1 modelOut
Failure to find variable name BeRefHHH                MainProcess[F]::setRunTimeVariable
Exiting from BeRefHHH not found                       ::main
Exit Stack:                                           ::main
::main                                                ::main
  MainProcess::InputModifications                     ::main
    MainProcess::setVariables                         ::main
      MainProcess[F]::setRunTimeVariable              ::main
```

Like most CombLayer error messages it is expected that you read them from top to bottom. So the first thing it tells you is that variable BeRefHHH does not exist in the model. Second that this is a fatal error and finally the calling stack that generated that error. If you are not debugging the code etc, then only concern yourself with the error/warning messages above the line *Exit Stack:*.

It is also possible to use strings and Vec3D objects as variables:

```
./ess -r -v BeRefTopRefMat Nickel \
         -v ABunkerQuakePtA0 'Vec3D(1200,191.2,0.0)' modelOut
```

will set the Top reflector material to *Nickel* and the start of the dilitation joint at 1200,191.2,0 (relative to target centre. Note that on bash you will need to hard quote (single quote) the Vec3D value or it will be split into commands.

Changing variables with XML file Create an XML file with the following content:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<metadata_entry>
  <Variables>
  <variable name="BeRefHeight" type="double">50</variable>
  <variable name="BeRefTopRefMat" type="string">Nickel</variable>
  </Variables>
</metadata_entry>
```

and generate the modified geometry:

```
./ess -r -x model.xml modelOut
```

All the variables can be exported in the XML file by running

```
./ess -r -X modelOut.xml modelOut
```

Note that the `-X` command will overwrite existing files.

The commands can be done combined simulantiously with both XML and command line and output.

```
./ess -r -X modelOut.xml -x model.xml -v BeRefHeight 80.0 modelOut
```

This will first read the model.xml file and change the variables. Second it will change `BeRefHeight` to 80, and finally write out the XML file of all variables to the file `modelOut.xml`.

Dealing with dependent variables If some other variables depend on the variable you are going to change, it can break the geometry. Consider, for instance, changing the BeRef radius from the baseline value of 34.3 to 40 cm:

```
./ess -r modelOut
```

this generates the 34.3 cm model[1] as shown in Fig. 6a.

Now let's set it to 40 cm:

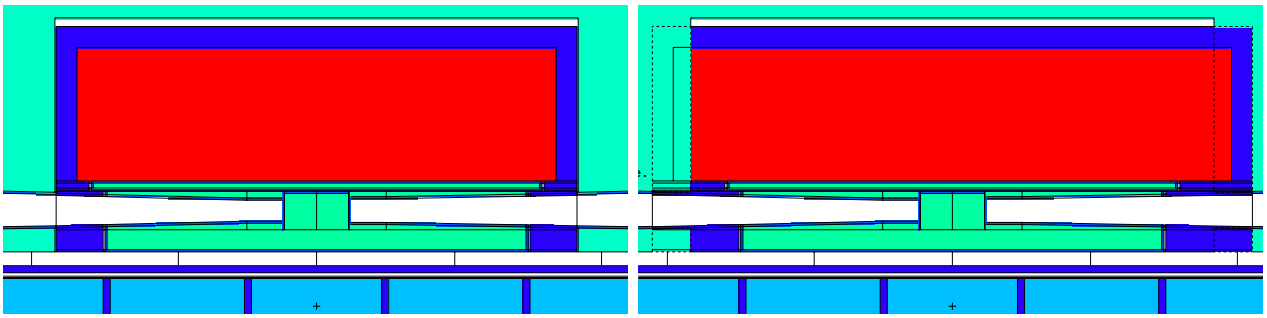
```
./ess -r -v BeRefRadius 40 modelOut
```

This geometry is shown in Fig. 6b and it is broken since now `BeRef` intersects with the Bulk component.

In order to find out which other variables depend on the given one (`BeRefRadius` in our case), find it in the variables setup in the C++ variable definition:

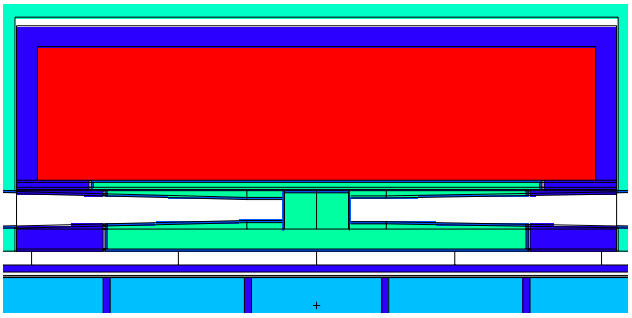
```
grep BeRefRadius Model/essBuild/*.cxx
Model/essBuild/essVariables.cxx:   Control.addVariable("BeRefRadius",34.3);
Model/essBuild/essVariables.cxx:   Control.addParse<double>("BulkRadius1",
                                                  "BeRefRadius+BeRefWallThick+0.2");
```

[1] 34.3 cm is the baseline BeRef radius in the commit `e47bf6d`.



(a) Baseline radius of 34.3 cm

(b) `BeRefRadius` increased to 40 cm without taking into account dependent variables, which produced geometric errors



(c) `BeRefRadius` increased to 40 cm taking into account dependent variables

Figure 6: Geometries with different BeRef radii

It means that we have to adjust the `BulkRadius1` variable accordingly. This variable depends upon both `BeRefRadius` and `BeRefWallThick`. Find out the `BeRefWallThick` value:

```
grep BeRefWallThick modelOut1.x
c BeRefWallThick 3
```

Therefore the `BulkRadius1` value must be $40 + 3 + 0.2 = 43.2$ cm:

```
./ess -r -v BeRefRadius 40 -v BulkRadius1 43.2 modelOut
```

which produces the correct geometry shown in Fig. 6c.

Important note Sometimes in the C++ variable definitions the dependence is not set explicitly, i.e. in our case `BulkRadius1` would be defined just by the value:

```
Model/essBuild/essVariables.cxx:    Control.addParse<double>("BulkRadius1",43.2);
```

In this case we have to inspect the geometry manually in order to find out which other variables we need to change to produce the correct input deck.

7.2 Variance reduction

7.2.1 Cell-based biasing

Cell-based weight window can be generated with the following arguments:

```
#! /bin/bash

./ess -r -defaultConfig Single ODIN -angle objAxis odinAxis 0 \
    -w \
    --weightEnergyType energy 0.1 0.95 1.0 0.85 10.0 0.5 100.0 0.4 5000.0 0.3 \
    --weightSource 'Vec3D(200.0,4.0,13.7)' \
    --weightPlane  'Vec3D(2800,0,0)' 'Vec3D(1,0,0)' \
    --weightObject G2BLineTop20          SS0 0.0 1.0 0.9 1.0 2.0 \
    --weightObject CBunkerWallMainWall1 TP0 0.0 1.0 0.9 1.0 2.0 \
    --weightObject CBunkerWallMainWall2 TP0 0.0 1.0 0.9 1.0 2.0 \
    --voidUnMask \
    TA
```

The purpose of the variance reduction is to change the weight in the cell by the following equation

$$w_{mod} = \text{scaleFactor} \times \frac{\exp(-\sigma \times \rho \times \text{densityFactor} \times r \times \text{rScale})}{(\text{rScale} \times r)^{\text{r2Power}}} \tag{1}$$

Note the repeate of rScale in the equation, thus effectively separating densityFactor from rScale. For each –weightObject the weight in the cells in modifed by w_{mod} in the equation above.

-defaultConfig Single ODIN These optional arguments build the ODIN beam line without building any other beamlines (see Fig. 7)..

-angle objAxis odinAxis 0 This rotates the model about the master z axis followed by around the master y axis such that the FixedComp link-point axis specified after objAxis is collinear with the x axis in the final output. In this example the FixedComp is the *odinAxis* and the link point is zero (which signifies to used the FixedComp origin and Y axis rather than a linkPoint). This rotation simplifies the weight window source plane setup for the current example and used here for illustration purpose.

-w should precede all weight-related arguments and sets up the model for variance analysis.

–weightEnergyType defines energy grid[2]. The following variants of syntax are possible:

1. If use the word *energy* then the format is energy followed by initial weight for that bin (in our example: for energies below 0.1 MeV the initial weight is 0.95; for energies between 0.1 and 1 MeV the initial weight is 0.85 etc).
2. If you drop the word *energy* then you just set the energy grid and all the initial weights are taken as 1.0
3. Alternatively you can use pre-defined keywords: `basic`, `high`, `mid` and `flat`[3].

–weightSource This argument defines a point in space. We can define as many as we want (by adding several `weightSource` arguments), but only those used with `weightObject` will be used. When referenced they will be called S0, S1, S2 ... etc based on the order they appear in the command line. This particular point is shown by a circle in the left side of Fig. 7a.

–weightPlane This argument defines a plane. Similarly to `weightSource`, we can define as many planes as we want, but only those used with `weightObject` will be used. When referenced they will be called P0, P1, P2 ... etc based on the order they appear in the command line. This particular plane is shown by a dashed vertical line in the right side of Fig. 7a.

–weightObject Define objects we would like to make cell-based variance reduction to.

[2]With the `wwg` card you can either use this expression or `--wwgE`

[3]Energy-weight binnings for these keywords are defined in `System/weights/WeightControl.cxx`

G2BLineTop20 Object name. It's also possible to specify range of objects within an object or cell name via `CellMap`.

SS0 This is a two part reference. The first letter (S) implies that the item is considered to emmit neutrons. The second characters imply that this occurs from the position of the first source point defined above.

energyCut=0.0 Energy cut. Defines min energy for variance reduction (i.e. no modification will occur to energy bins below this value). However, if the number is negative, then if defines a -ve maximum energy past which no variance modification will be made.

scaleFactor=1.0 Linear scale factor in equation **??** It's transport to go from *a* cell to another cell. Scale factor is taken out in the mesh based variance reduction because it is identical in effect to the command [-wwgNorm scaleFactor 1.0]

densityFactor=0.9 Density scale factor. Normally we set it a bit less than one to make it easier to do transport through thick layers

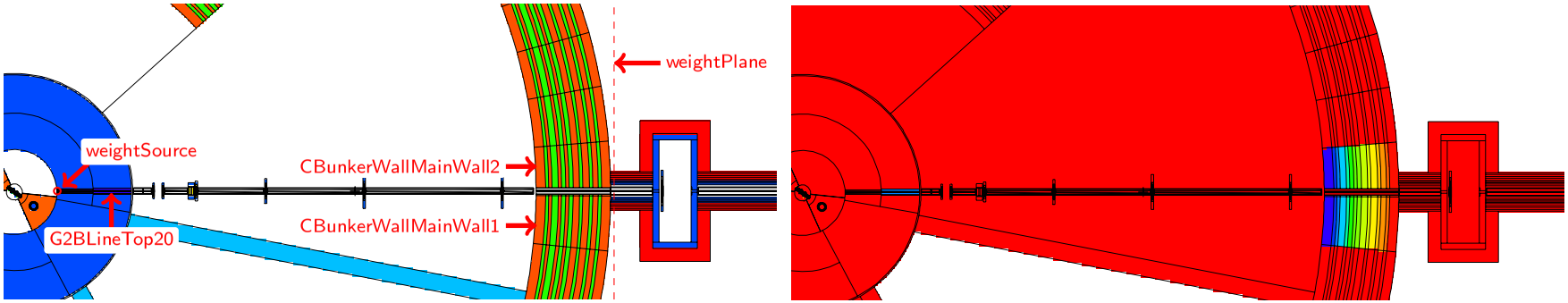
rScale=1.0 Length is adjusted to a different scale effect

r2Pow=2.0 Scale for the length exponent in the distance effect. It should be increased in high absorbing regions and decreased in regions that are effectively in a forward going direction for the higher energy scattering (effectively mimicking forward angle scattering)

–voidUnMask The unpopulated world is made of sphere of importance 0 both inside and out. All objects that are places in the world must go in the inner spherical volume and they (unless specified) will have importance 1.0. In the case that the tally extends into this inner region sphere (e.g. for a dose after a shielding wall) must have either (a) a void volume added to the model or (b) the inner part of the world shere set to importance zero. This flag does option (b) [This flag may also be used to allow cross talk between two beamlines for example]

Notes

- In order to set up biasing, you need at least one source, if you one or more tally (adjoint) points then you must have at least one source.
- Source can be defined either as a point (via **–weightSource**) or a plane (via **–weightPlane**). In the example above we defined point source. However the defined sources and planes do nothing until you use them in **–weightObject**.
- General rule to get the things into work: try different setups and compare results. In particular, higher energy transport is forward going and it is easy to over weight the splitting when particle would predominately scatter in a forward direction. This can be overcome by either reducing the r^2 power or decreasing the density. Similarly, resonance streaming dominates thick shielding transport but the code does not account for that, so to approximate the lower attenuation coefficient the density can be reduced using the density factor. Finally, for H_2 systems (e.g. polyethelene shielding) the high energy lose term for the hydrogen is not correctly weighted (CombLayer uses single group approximation) and additional r^2 or Rlength scaling is worth cosidering.



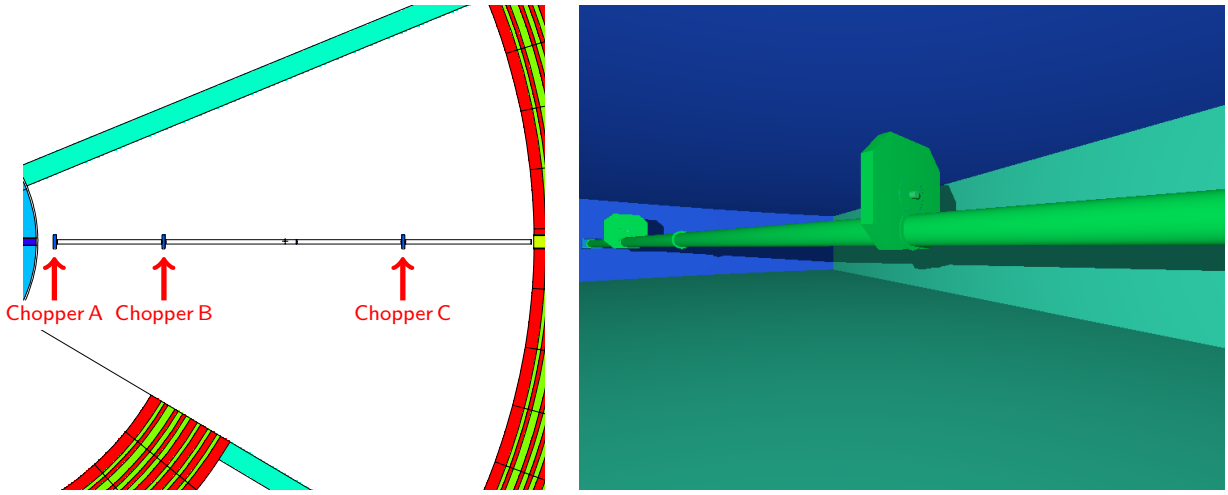
(a) Horizontal view: geometry

(b) Horizontal view: wwn

(c) Vertical view: geometry

(d) Vertical view: wwn

Figure 7: Cell-based weight window



(a) MCNPX: horizontal cut through the beam line

(b) Same geometry rendered with POV-Ray

Figure 8: Geometry of the BIFROST beam line inside Bunker

7.3 Activation

The activation calculations are performed in several steps as described in the sections below:

1. Prepare geometry.
2. Calculate neutron fluxes using MCNP 6.
   These first two runs are completely independent, but they must use the same model. Normally the difference between them is that the second run has the energy cut, while the first — not. In principle we do not need the energy cut, but it saves time.
3. Generate the htape files with MCNPX.

7.3.1 Geometry

In this particular example we are going to look at a chopper of the BIRFOST beam line: how active will the chopper beam port become when we try to take it off?[4] In other words, we will calculate residual dose and gamma spectra.

First, we will build the BIFROST geometry and rotate it around its axis[5]:

```
./ess -r -defaultConfig Single BIFROST -angle objAxis bifrostAxis 0 \
      -v bifrostStopPoint 3 \
      ChopA
```

The beam line geometry is shown in Fig. 8.

7.3.2 Neutron fluxes

Now we need to set up neutron flux f4 tallies in all non-void cells of ports A and B of the chopper A (see geometry in Fig. 9):

[4] I.e. the rotors and motors are lifted out (with a crane) and we are left with the chopper housing.
[5] The bifrostStopPoint argument is described in Sec. 7.4.1 on page 25.

```
./ess -r \
    -defaultConfig Single BIFROST \
    -angle objAxis bifrostAxis 0 \
    -v bifrostStopPoint 3 \
    -T flux n allNonVoid bifrostChopperAIPortA \
    -T flux n allNonVoid bifrostChopperAIPortB \
    -cinder -TMod single -4 \
    -n 80000 \
   ChopA
```

The `-cinder` argument does two things:

- adds the `histp` card with all the cells listed in both `f4` tallies.
- creates a file called `materials` with material definitions later used by whom?

The default energy bins generated by the `-T flux` cards already match the `CINDER` binning.

The `-TMod single -4` arguments remove brackets around the cells used in all `f4` tallies. `-4` means 'all `f4` tallies', while `4` would mean 'only `f4` tally'.

Of course, for `MCNPX` simulations this flag is needed: `-mcnp 10`. You need to have neutron fluxes with good statistics, therefore it is advised to use variance reduction (see Sec. 7.2 on page 17).

> You need to fix the `MCNP source.F` file to correct velocity and cell finding error and allow to start SSR not at a surface.
>
> SA puts bounding box of neutrons to come in out of a beam port and then he restarts those neutrons within this box. Convenient thing is that in the new geometry the surface numbers may change (as they always do with CombLayer), but this fix allows SSR start particles from a bounding box not caring about surface numbers[6].

7.3.3 htape

Second run generates the `htape` file:

```
./ess -r \
    -mcnp 10 \
    -C 0.1 \
    -defaultConfig Single BIFROST \
    -angle objAxis bifrostAxis 0 \
    -v bifrostStopPoint 3 \
    -T flux n allNonVoid bifrostChopperAIPortA \
    -T flux n allNonVoid bifrostChopperAIPortB \
    -cinder -TMod single -4 \
    -n 80000 \
   ChopHA
```

You do not need it if you do not care about spallation reactions.

The energy cut `-C 0.1` needs in order to speed up since we get particles only from histp. `-mcnp 10` is needed since we are going to generate `htape` with `MCNPX`.

[6] I.e. the SSW and SSR cards do not have to have the same surface numbers.

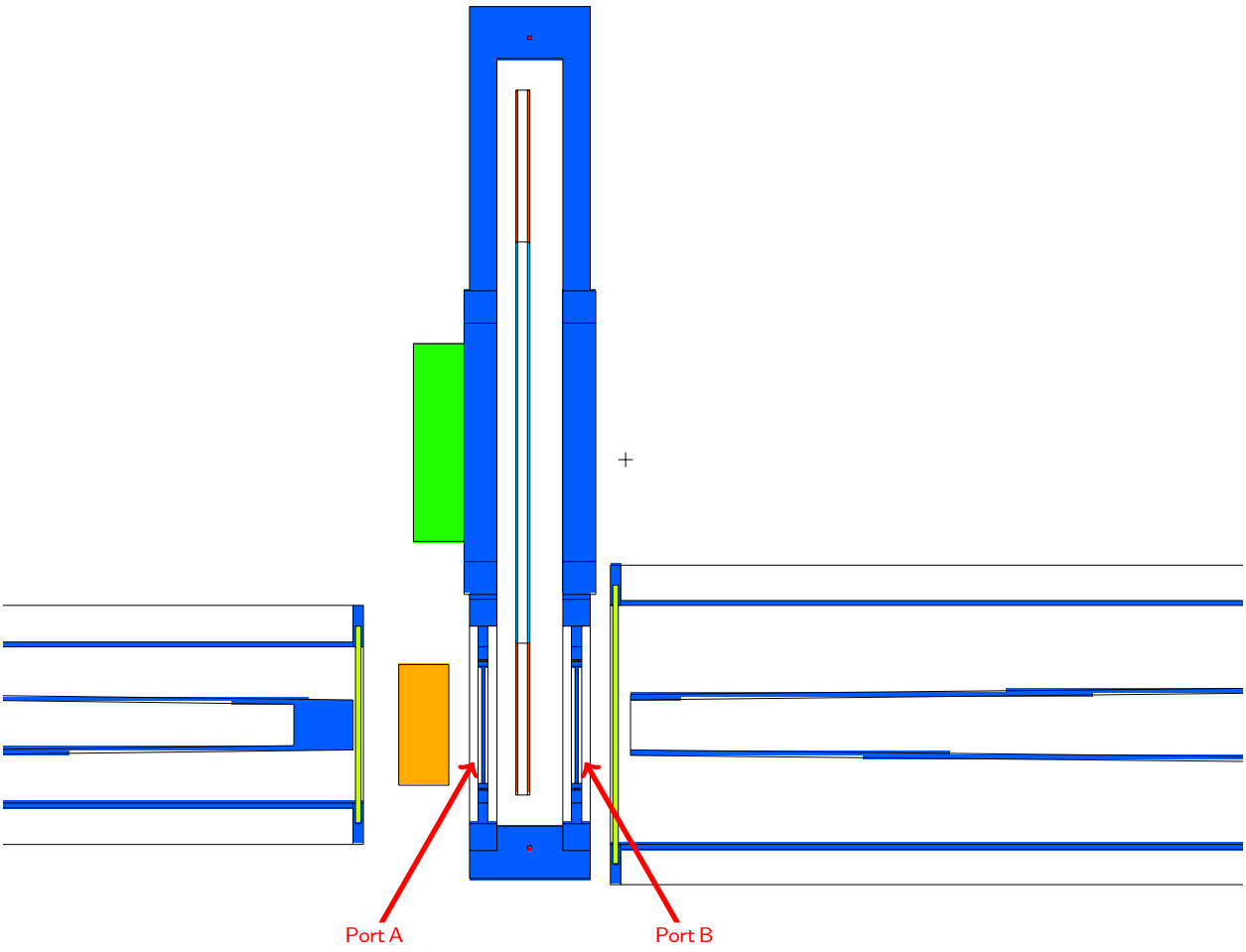


Figure 9: Geometry of the BIFROST chopper A

7.3.4 Activation script

Now we have to prepare the activation script input file. Its syntax is similar to the CINDER one (inpact), but much more forgiving. We call it bifrostStop01:

```
title_lines
  BIFROST chopper Windows
  test

files
  mat_file      ChopA1.o
  mcnpx_outp    ChopA*.o
  mcnpx_histp   ChopA1*.h
  htape_exe     /path/to/htape3x
  cinder_exe    /path/to/cinder
  tabcode_exe   /path/to/tabcode
  library       /path/to/c90lib0742

run_options
  dname     Cell
  dcounter 1

cinder_options
  tst       1e-5
  signif    1e-16
  kchn      0
  klib      0
  nfe       3
  nosame    0
  russ      1

normalization
  snorm   2.66e10
  hnorm   1.2e9

history
1 1.0E+00 5000 h
4 0.0E+00 1.0 h -1.0 d  -7.0 d -30.0 d -1.0 y


cell_list
  A tally ChopA1.x 4
  A tally ChopA1.x 14
```

Lines can be concatenated by backslash. The order of keywords in the given section does not matter.

title_lines as many lines as necessary are possible. Everything longer 72 characters is chopped.

mat_file materials. Normally one of the MCNP output files

mcnpx_outp list of all the files you wish to sum together. Wild-masks and directories are supported.

mcnpx_histp list of `histp` files

htape_exe htape3x

cinder_exe cinder

tabcode_exe tabcode

library CINDER libs

7.4 Geometry models

This section describes features specific for some CombLayer models.

7.4.1 ESS

Beam lines

StopPoint variable The `StopPoint` variable stops building a beam line at a given point:

0 builds the whole beam line

1 builds nothing more than a bit up to the 5.5 m mark

2 builds everything in the bunker but not through the bunker wall

3 builds up to the end of the Bunker

4 builds up to the cave

5 bulds everything beyond the cave (object where the sample/detector are located)

If you use the `essBeamLine` executable then you can also give the start point and then you will only get a fragment of your instrument. Obviously, you would get nothing if the start and end point are the same. syntax of ./essBeamline?