



Philosophy, Principle, and Method for the CombLayer

Stuart Ansell

European Spallation Source, Lund, Sweden.

December 8, 2015

Icons are added to slide to indicate issues

- Unexpected/Dangerous code



- Legacy Code



- Philosophy Driven decisions



MCNP(X) is a geometry/physics/variance reduced discrete Monte-Carlo transport code.

Advantages

- World is described in complete volume format using infinite quadratic surfaces.
- Almost every default is *best possible result* without modification.
- MCNP(X) is tally based – meaning that MCNP(X) does not calculate stuff you don't ask for.

Consequences

- Complete volume description allows fast simulation – but only with small cells [small \implies small number of surfaces]
- CVG implies that changing one cell requires the change of **ALL** cells that it exists directly within
- Cell-to-Cell (C2C) transport is dominated by surface crossing
- Tallies and variance reduction have to be incorporated in build

MCNP(X) is an unforgiving code

MCNP adds a huge number of things to help build geometry.

BUT All of the following normally result in runtime penalty

- Complementary cells
- Universes require full computation of objects within
- Transform cards / Lattice cards
- Macrobodyes
- Boolean invariances

MCNP(X) allows the definition of quadratic surface types
To define a surface the input file has a line :

```
1 | IDNumber typeID [values...]
```

Examples

```
2 | 57 px 3.4           A plane on the x=3.4 surface  
3 | 1782 p 0.5 0.5 0 3.4   A plane on the x=y plane  
4 |  
5 | 983 c/x 1.2 3.4 7.0   A cylinder along x axis at y=1.2 and  
6 |                       z=3.4 radius 7.0
```

MCNP(X) objects are constructed from a set of boolean operations on quadratic surface

Each object is ALSO numbered (!) and using signed surface numbers (to mean true/false).

Intersection (i.e. all have to be true for a point to be in the object) is denoted by a space

Union (i.e. only one has to be true for a point to be in the object) is denoted by a colon (:)

```
1 | objectID MatID Density [surfaces ]
```

Making a simple cube of size 8cm in each direction :

Examples

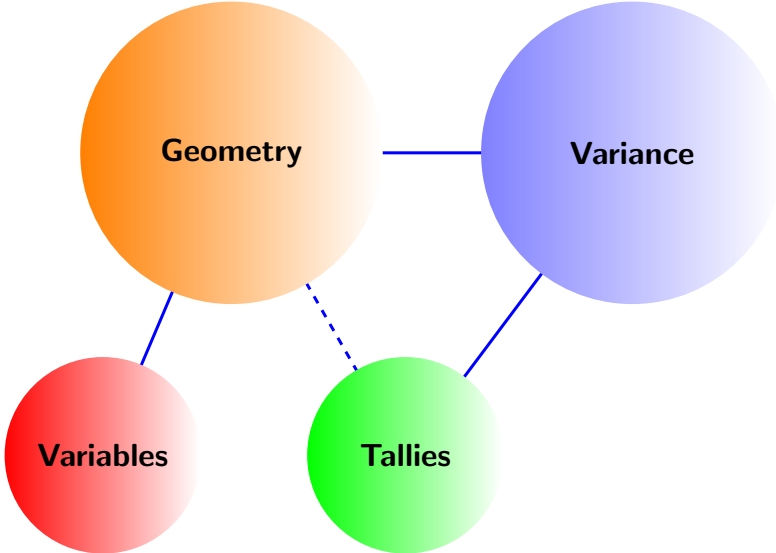
```
8 11 px 4.0
9 12 px -4.0
10 13 py -4.0
11 14 py -4.0
12 15 pz -4.0
13 16 pz -4.0
14
15 Object with material 4 / density 0.051674 4 0.05 11 -12 13 -14 15 -16
```

These little MCNP units are the main sub-building block of CombLayer

Placing that cube in a sphere of radius 45 cm. **Examples**

```
18 25 so 45.0
19
20  Object with material 4 / density 0.05
21 1674 4 0.05 11 -12 13 -14 15 -16
22 1872 9 0.073 -25 (-11:12:-13:14:-15:16)
```

Note the boolean negation of the original cube in the sphere object.



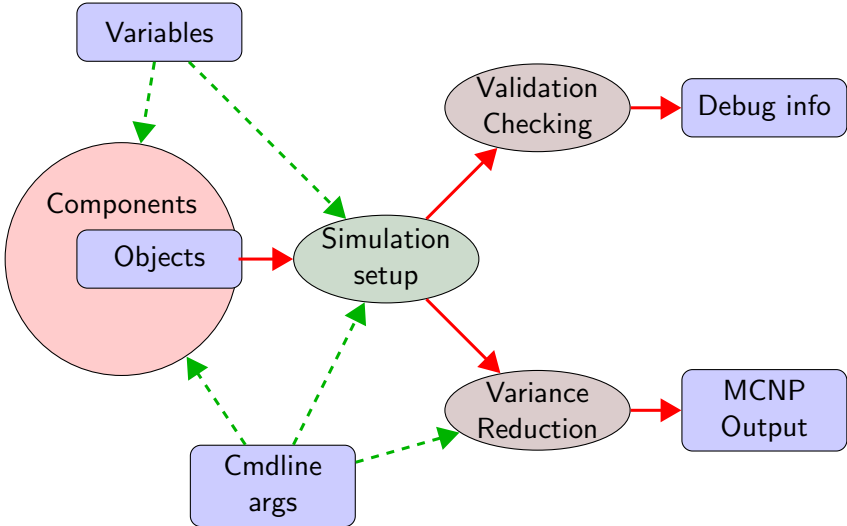
What does CombLayer provide

- Variable system [fully Turing Complete]
- Boolean level object handler
- Higher level component system
- Variance reduction system
- Error detection

What is NOT in CombLayer

- No minimization routines for variables
- No tally data analysis
- No error correction – immediate failure on problem

CombLayer Process



Object Section

Boolean
Rules

index lists

Surface Section

Quadratic
types

IDnum : type : values

Remainder Section

weights

IDnum : type : values

Materials

IDnum : type : values
modification lines

Physics

type : flagNumbers

Source

type : values
subtype : values

tallies

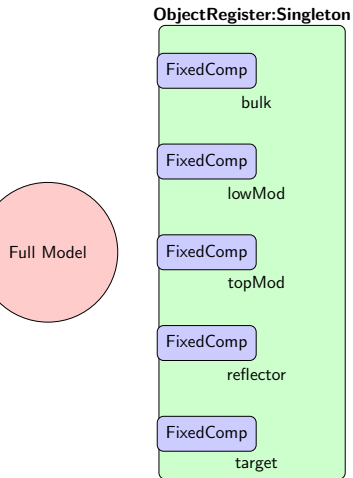
type : values

```
24 221 5 0.0582256 -244 245 ( -305 : 306 : 204 : 206 : ( -212 209 )
25      223 : -220 ) -308 307 -215 -217 tmp=1.72346844e-09
26 222 5 0.0582256 3 -245 ( -305 : 306 : 205 : 206 : ( -214 211 ) )
27      : -222 ) -308 307 -216 -217 tmp=1.72346844e-09
28
29 49 c/z 59.35095759 94.98138677 20
30 150 c/z 59.35095759 94.98138677 22
31 151 p 0.891006524 -0.4539905 0.0 9.26144319
32 152 p 0.891006524 -0.4539905 0.0 10.2614432
33 142 gq 1 1 23.01803079 0 0 0 -118.7019152 -189.9627735
34 0
      -4135.732457
```

Arrh.....

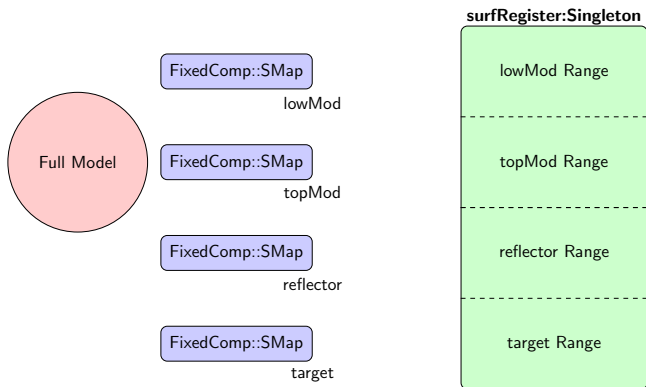
- Consider the model as a whole set of mini-models
- Provide simple interaction between the models
- Avoid the need to use global identifier numbers
- Help out on the service level (variance/tallies)

Geometry: Objects [After Construction]



- Each component is a separate and independent
- Components know everything about themselves (materials / surfaces / position etc.) and **nothing** about anything else
- Components all stored in a *singleton* objectRegister.

Geometry: Surfaces [After Construction]



LOCAL surface map for each Component with global surface access.

std::cout output is expected to be replaced with a managed stream:

ELog::EM

std::endl is replaced by of choice of

- ELog::endDiag
- ELog::endCrit
- ELog::endErr
- ELog::endTrace
- ELog::endBasic

Example:

```
1 |   for(size_t i=0;i<10;i++)
2 |       ELog::EM<<"test " <<i<<" " <<sin(i*3.0)<<ELog::endDiag;
```

Adding instance of `ELog::RegMethod` to all methods that

- 1 throw exceptions
- 2 write output
- 3 call methods that do 1 or 2.

`RegMethod`'s main constructor that takes two strings. Typically I use that for the class name and the method name.

Example:

```
3 void BigCave::calculate ()
4 {
5     ELog::RegMethod RegA(" BigCave" ," calculate" );
6     ELog::EM<<"Some info"<<ELog::endDiag;
7     // ...
8     ELog::EM<<"HARD break"<<ELog::endErr;
9 }
```

Output:

Some info

HARD break

::main

makeEss::build

CSpec::createAll

BigCave::calculate

Bunker::createMainWall

Bunker::createMainWall

Unsurprisingly CombLayer has a Vec3D class. However, it is unusual.

- Vec3D provides typical $+$, $-$, $*$, $/$ and $[]$ operators
- Vec3D is a valid variable type
- It provides scalar interaction **BUT ONLY** forwards:

This is valid:

```
1 |
2 |  const Vec3D Out=Origin+X*7.9;    // VALID
3 |  const Vec3D OutX=Origin+7.9*X;   // ERROR
```

The normal methods stuff you would expect to see

- *unit()* : return a unit vector
- *rebase(const Vec3D& A, const Vec3D& B, const Vec3D& C)* : express vector in basis set A,B,C
- *Distance(const Vec3D& A)* : distance to vector
- *dotProd(const Vec3D& A)* : dot product

Working with vectors involves rotation: to avoid gimbal-lock quaternion rotation is preferred over Matrix/Euler rotation.

Rotational Quaternions can be created/used directly as :

```
5  
6   Geometry::Vec3D axis(1,0,0);  
7  
8   // apply rotation about Z axis:  
9   const Geometry::Quaternion Qz=  
10      Geometry::Quaternion::calcQRotDeg(rotAngle,Z);  
11  
12  Qz.rotate(axis);
```


Most of the time – FixedComp will deal with it for you because you will be rotating a whole object

```
14  
15 FixedComp::applyShift(xStep, yStep, zStep);  
16 FixedComp::applyAngleRotate(xyAngle, zAngle);
```

Object component is a single class that provides a mechanism to produce surfaces and single material cells based on the values of variables.

Examples:

- Moderator
- Jaws
- Sample
- Detector

Composite component is a single class holds a collection of Object components making a more complex object but for some instances wishes to be treated as a single object

Examples:

- Target
- Bunker
- Cave
- Chopper system

A basic geometry object normally has 2-3 of the public inheritances from the namespace **attachSupport** ¹

We are going to cover:

- 1 FixedComp / FixedGroup **Mandatory** ²
- 2 ContainedComp / ContainedGroup ²
- 3 LayerComp
- 4 CellMap
- 5 SurfMap

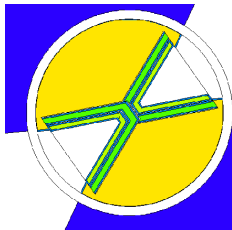
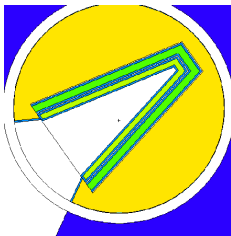
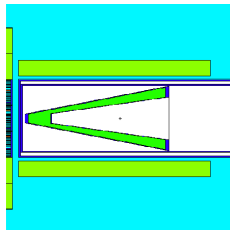
¹Files in System/attachSupportInc

² This class should not be virtually inherited.

CombLayer Component Construction

Models are made of components:

- Local parameters e.g. number of layers
- Containers that allow insertion
- Link points that allow joining



Same object

Most geometry components inherit from FixedComp

It Provides:

- A unique name [keyName]
- Origin for the object
- X,Y,Z basis for object
- A local name-register for surfaces
- Stores/Accesses link points for the object



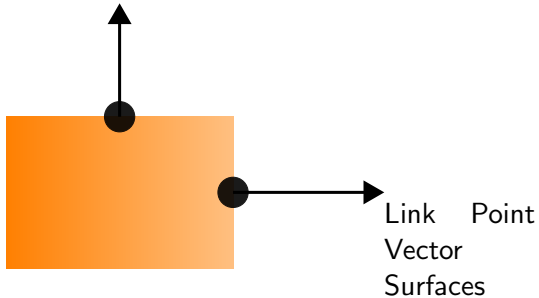
Construction of a FixedComp is done as follows:

```
36 class MyBox : public attachSystem::FixedComp
37 {
38     const int  boxIndex;          ///< Index of surface offset
39     int  cellIndex;              ///< Cell index
40
41     public:
42
43     MyBox(const std::string&);
44 };
45
46 MyBox::MyBox(const std::string& Key) :
47     attachSystem::FixedComp(Key,0),
48     boxIndex(ModelSupport::objectRegister::Instance().cell(Key)),
49     cellIndex(boxIndex+1)
50 {}
```



Typical FixedComp object looks like:

```
2
3 class MyBox : public attachSystem :: FixedComp ,
4   public attachSystem :: ContainedComp
5 {
6   private :
7
8     const int  boxIndex;          ///< Index of surface offset
9     int  cellIndex;              ///< Cell index
10
11   void createUnitVector(const attachSystem::FixedComp&,
12                         const long int);
13
14   void createSurfaces();
15   void createLinks();
16   void createObjects(Simulation &);
17
18   public :
19
20     void createAll (Simulation&,
21                   const attachSystem::FixedComp&,
22                   const long int);
23 }
```

- Objects are constructed with links
- Other objects are realised relative to these linkages
- The outer Link Surfaces are designated the *boundary*
- Boundaries are checked after construction to allow displacement construction

Definition of a link-point

- A point in space (`Geometry::Vec3D`)
- An axis direction in space (`Geometry::Vec3D`)
- A surface(s) rule
- **An extra common surface if required**

Adding/Defining Link-Points

- The number of expected link-points is defined in the constructor
- Numbers 0-9 are designated outgoing **convention**.
- Numbers 10-19 are designated in-going **convention**.

```
25
26 class MyBox : public attachSystem :: FixedComp
27 {
28     private:
29
30     // this is in most FixedComp derived classes
31     void creatLinks();
32 };
33
34 MyBox::MyBox(const std::string& Key ) :
35     attachSystem::FixedComp(Key, \tcr{\bf 6}),
36     boxIndex(ModelSupport::objectRegister::Instance().cell(Key)),
37     cellIndex(boxIndex+1)
38 {}
```

Typical usage - a box with 6 sides (hence 6 link points)

Typically I add link points in linkCreate. Not always !!

```
39 |
40 | void
41 | MyBox::createLinks ()
42 | {
43 |     FixedComp::setConnect (0, Origin+Y*(width/2.0), Y);
44 |     FixedComp::setLinkSurf (0, -SMap.realSurf (boxIndex+1));
45 | }
```

- setConnect :: Defines a point and a direction (Y).
- setLinkSurf :: defines the surface LEAVING the box
- **A real surface is required**

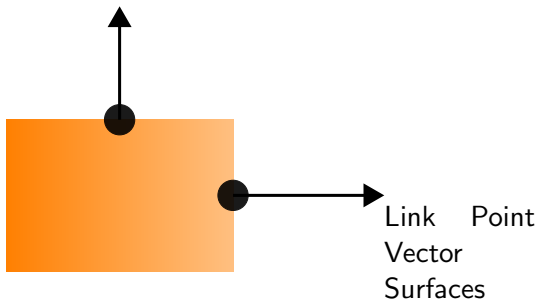
How to use link-Points [for Geometry]

The commonest method is simple orientational construction:

```
46  
47 void  
48 MyBox::createUnitVector(const attachSystem::FixedComp& FC,  
49                        const long int sideIndex)  
50 {  
51     FixedComp::createUnitVector(FC, sideIndex);  
52     FixedComp::applyShift(xStep, yStep, zStep);  
53     FixedComp::applyAngleRotate(xyAngle, zAngle);  
54     return;  
55 }
```

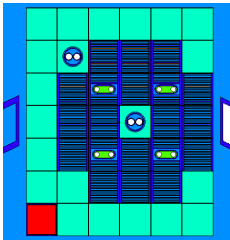
Creates a Vec3D Origin to be at link-Point [sideIndex]

CombLayer Object Construction

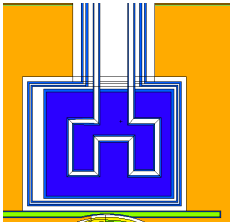


- Objects are constructed with links
- Other objects are realised relative to these linkages
- The outer Link Surfaces are designated the *boundary*
- Boundaries are check after construction to allow displacement construction

CombLayer Component Construction



- Simple connection grid joining at common side surfaces
- Track boundary e.g. for pipe work



Many geometry components inherit from
ContainedComp/ContainedGroup

It Provides:

- A description of the outer boundary
- A description of the inner boundary [optional]
- An approximation to the boundary assuming this is the complete object

Usage:

- Inherit into class
- Either register before/after which other objects this object is in
- Define an outerSurface using the command

```
1
2  class MyBox : public attachSystem::ContainedComp
3      public attachSystem::FixedComp
4  {
5      void createObjects(Simulation&);
6  };
7
8  void MyBox::createObjects(Simulation& System)
9  {
10     std::string Out;
11     // ...
12     Out=ModelSupport::getComposite(SMap, bnkIndex ,
13         " 1 -17 4 -14 5 -6 ");
14     System.addCell(cellIndex++, matID, 0.0, Out);
15 }
```



Models are constructed into a data-class called Simulation.

- Simulation is a base class for the output type required [PHITS/MCNP etc]
- In principle you can have two but... **don't**
- Joins Physics - Geometry - Tallies
- Works on a *push-to / pull-from* model

Note: In 99% of the code, I have used this :

```
1 |  
2 | void MyBox::someFunction(Simulation& System)
```



Example use [Adding a cell]:

```
4 |  
5 | System.addCell(cellIndex++,matID,300.0," 2 -3 -4 5 6 ");
```

This adds a cell [id number cellIndex] with material type matID and temperature 300K, using the MCNP surface logical format string.



Example use [find cell

```
7  
8 // Find a cell containing a point:  
9  
10 MonteCarlo::Object* OCell=  
11   System.findCell(Geometry::Vec3D(4,5,6),0);
```



You also used it to do things with everything in the model:

```
15  
16 // Global surface replacement:  
17 // BE VERY CAREFUL DOING THIS::  
18 System.substituteAllSurface(1082,283);
```

Starting a project

- 1 Copy Main/pipe.cxx to Main/newProject.cxx
- 2 create a directories in Model/projectBuild and Model/projectBuildInc
- 3 Edit CMake.pl
 - Add **newProject** to masterprog
 - Copy the line `$gM-;addDepUnit(" pipe" , ...`
 - Change pipeBuild to projectBuild

Starting a project (part2)

- 1 Copy Model/pipe/*.cxx to Model/project/
- 2 Copy Model/pipeInc/*.h to Model/projectInc/
- 3 Change each item of pipe to project

Consider the Main :