FOOT simulations with FLUKA in SHOE newgeom branch

MILANO TEAM

SEPTEMBER 2019

Introduction

Managing FLUKA simulation in SHOE

- Main steps:
- shoe/build is the working directory for both reconstruction and simulation. To run the simulation the user must work in shoe/build/Simulation
- Prepare input and geometry files
- Build the FLUKA executable using the FOOT user routines
- Run the simulation
- Collect the output files and producing the ROOT treeSHOE has the tools which allow to modify/build the input/geometry etc.

Preparing geometry & input files: MakeGeo

MakeGeo

- In shoe/build/Simulation are stored the simulation files. Input and geometry files, can be built according to your purposes by means of makegeo
- In shoe/Simulation/MakeGeo.cxx are contained all the instruction to produce these files
- When you compile the code (*make* in shoe/build/Simulation folder) the executable makegeo is produced in shoe/build/bin
- You can execute it in the shoe/build/Simulation folder (*../bin/makegeo*) to produce the simulation files
- But what does makegeo do? It is based on shoe libraries (shoe/libs/scr/*), it reads parameter files and produces 3 files needed to run the simulation:
  - foot.inp → input file (beam, materials, etc) → Opens and modifies the existing foot.inp
  - foot.geo → geometry file → Creates both files from the scratches
  - parameters.inc → include file of parameters needed by the user routines → Creates both files from the scratches

- DISCLAIMER: no instruction about libraries management/modification will be given in this tutorial

Main parameters files for MakeGeo

ASCII files:

- shoe/build/Reconstruction/level0/config/FootGlobal.par
  it allows choose which detectors to simulate by putting yes or no in a list.
- In shoe/build/Reconstruction/level0/geomaps there are:
  - FOOT_geo.map which contains the positions and rotation angles in global coordinates of all FOOT detectors and magnets
  - TA*detector.map which contain, for each single detector (or magnet system), the relative coordinates and rotation angle of every element composing the detector itself, together with the material description. TAGdetector.map contains infos about target and beam
- These files also allow to choose and address the proper map of magnetic file, which is contained in shoe/build/Reconstruction/fullrec/data
- An easy and quick access to these folders (config, geomaps and data), and so to their files, is provided by logical links in shoe/build/Simulation

```
0 lrwxr-xr-x   1 serena  staff    74B 22 Set 17:13 config -> /Users/serena/Lavoro/FOOT/newgeom/shoe/build/Reconstruction/level0/config/
0 lrwxr-xr-x   1 serena  staff    73B 22 Set 17:13 data -> /Users/serena/Lavoro/FOOT/newgeom/shoe/build/Reconstruction/fullrec/data/
0 lrwxr-xr-x   1 serena  staff    75B 22 Set 17:13 geomaps -> /Users/serena/Lavoro/FOOT/newgeom/shoe/build/Reconstruction/level0/geomaps/
```

- <u>There is no need to recompile shoe after the modification of parameter files</u>

MakeGeo - reading par files (I)

MakeGeo.cxx

```
GlobalPar::Instance("FootGlobal.par");
GlobalPar::GetPar()->Print();
```

```
TAGroot* fTAGroot = new TAGroot();
TAGmaterials* fTAGmat = new TAGmaterials();

TAGgeoTrafo geoTrafo;

// GlobalFootGeo footGeo;

TADIparGeo* diGeo = new TADIparGeo();      Dipoles
TASTparGeo* stcGeo = new TASTparGeo();     Start counter
TABMparGeo* bmGeo = new TABMparGeo();      Beam monitor
TAVTparGeo* vtxGeo = new TAVTparGeo();     Vertex
TAITparGeo* itrGeo = new TAITparGeo();     Inner tracker
TAMSDparGeo* msdGeo = new TAMSDparGeo();   Microstrips
TATWparGeo* twGeo = new TATWparGeo();      Scintillator
TACAparGeo* caGeo = new TACAparGeo();      Calorimeter
TAGparGeo* generalGeo = new TAGparGeo();
```

FootGlobal.par

```
......

IncludeDI:                n
IncludeST:                y
IncludeBM:                y
IncludeIR:                n
IncludeTG:                n
IncludeVertex:            y
IncludeInnerTracker:      n
IncludeMSD:               n
IncludeTW:                y
IncludeCA:                y

.......
```

You can choose which detectors simulate by putting yes or no in shoe/build/Reconstruction/level0/config/FootGlobal.par
No need to recompile.

Creation of objects for materials (TAGmaterials), geo tansformations (TAGgeoTrafo), detectors (TASTparGeo, TABMparGeo, TAVTparGeo, TAITparGeo, TAMSDparGeo, TATWparGeo, TACAparGeo, TADIparGeo) and other general infos (TAGparGeo for beam, target, standard geometry regions etc).

FootGlobal.par

```
FLUKA version: pro
```

You can also change declare the FLUKA version (pro/dev) you are planning to use (reserved to developers)

MakeGeo - reading par files (II)

```
TString parFileName;
geoTrafo.FromFile("./geomaps/FOOT_geo.map");

// read geomap files
stcGeo->FromFile();
bmGeo->FromFile();
vtxGeo->SetMcFlag();
vtxGeo->FromFile();
itrGeo->FromFile();
msdGeo->FromFile();
diGeo->FromFile();
twGeo->FromFile();
caGeo->FromFile();
generalGeo->FromFile();
```

MakeGeo.cxx

The parameter files of each detector (shoe/build/Reconstruction/level0/geomaps/TA*detector.map) are read as well as the FOOT_geo.map, which contains the positions in global coordinates.

Example: TAVTdetector.map

```
Sensors:            4

// M28
TypeName:          "M28"
TypeNumber:         0
PixelsNx:          960
PixelsNy:          928
PitchX:          0.00207
PitchY:          0.00207
TotalSizeX:      2.0240   TotalSizeY:    2.27100  TotalSizeZ:     0.0050
EpiSizeX:        1.9872   EpiSizeY:      1.92096  EpiSizeZ:       0.0014
EpiOffsetX:      0.0356   EpiOffsetY:    0.32800  EpiOffsetZ:     0.0000
EpiMat:           "Si"
EpiMatDensity:   2.329
EpiMatExc:       174.5e-6
PixThickness:    0.00064
PixMat:         "SiO2/Al"
PixMatDensities: "2.65/2.70"
PixMatProp:     "0.89/0.11"
PixMatDensity:     2.3
SupportInfo:        0

// -*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*- Plane 1  -*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-
Inputs:             1
Type:               0
PositionsX:       0.0000  PositionsY:    0.0000  PositionsZ:     -0.92
Tilt1:            0.0000  Tilt2:         0.000   Tilt3:           0.000
AlignementU:      0.000
AlignementV:      0.000
AlignementTilt:   0.000

.....
```

FOOT_geo.map

```
// X,Y,Z and angles
StartBaseName: "ST"
StartPosX: 0. StartPosY: 0. StartPosZ: -29.
StartAngX: 0. StartAngX: 0. StartAngX: 0.

TargetBaseName: "TG"
TargetPosX: 0. TargetPosY: 0. TargetPosZ: 0.
TargetAngX: 0. TargetAngX: 0. TargetAngX: 0.

BmBaseName: "BM"
BmPosX: 0. BmPosY: 0. BmPosZ: -14.
BmAngX: 0. BmAng:Y 0. BmAngZ:   0.

VertexBaseName: "VT"
VertexPosX: 0. VertexPosY:   0. VertexPosZ: 1.5
VertexAngX: 0. VertexAngY:   0. VertexAngZ: 0.

MagnetsBaseName: "DI"
MagnetsPosX: 0. MagnetsPosY:   0. MagnetsPosZ: 16.5
MagnetsAngX: 0. MagnetsAngY:   0. MagnetsAngZ: 0.

....
```

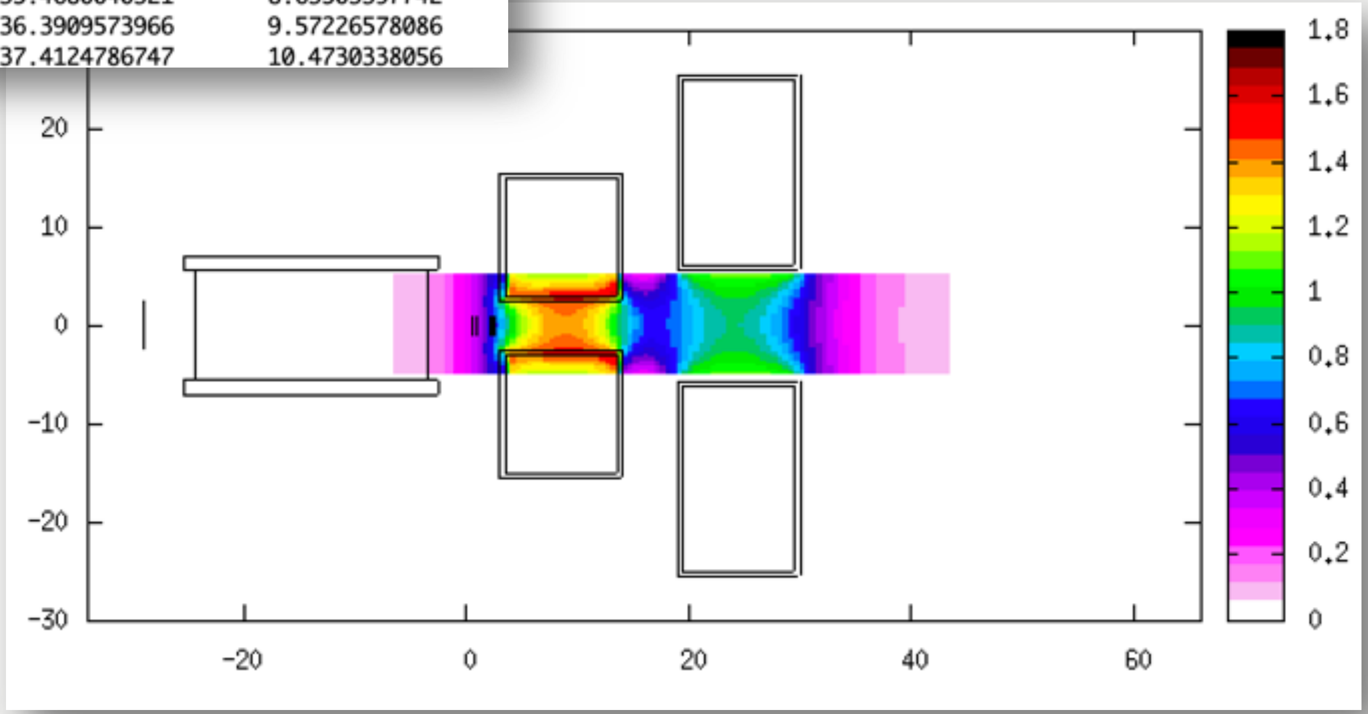
Positions, dimensions, material etc can be modified in this files. **No need to recompile the code after modifications.**

Magnetic field

x	y	z	Bx	By	Bz
88641 21 21 201					
-5.00000000000	-5.00000000000	-50.0000000000	1.36243669343	-30.6770814637	0.459840156935E-16
-5.00000000000	-5.00000000000	-49.5000000000	1.35464451635	-30.8022112988	0.528218691543
-5.00000000000	-5.00000000000	-49.0000000000	1.34685233926	-30.9273411338	1.05643738309
-5.00000000000	-5.00000000000	-48.5000000000	1.35906061221	-31.0238462683	1.64156333646
-5.00000000000	-5.00000000000	-48.0000000000	1.40067845000	-31.2369115992	2.25326833731
-5.00000000000	-5.00000000000	-47.5000000000	1.43021238978	-31.4820544068	2.78929569702
-5.00000000000	-5.00000000000	-47.0000000000	1.45974632956	-31.7271972143	3.32532305672
-5.00000000000	-5.00000000000	-46.5000000000	1.48928026933	-31.9723400218	3.86135041643
-5.00000000000	-5.00000000000	-46.0000000000	1.56924075328	-32.2800339546	4.50091052135
-5.00000000000	-5.00000000000	-45.5000000000	1.65712860643	-32.6735541981	5.15804511664
-5.00000000000	-5.00000000000	-45.0000000000	1.71563615145	-33.1755302923	5.73249807025
-5.00000000000	-5.00000000000	-44.5000000000	1.80905040239	-33.6194597825	6.32749263121
-5.00000000000	-5.00000000000	-44.0000000000	1.93494986313	-34.1492021605	7.15672152936
-5.00000000000	-5.00000000000	-43.5000000000	2.04046917779	-34.7722498674	7.87790619204
-5.00000000000	-5.00000000000	-43.0000000000	2.20446996652	-35.4680040321	8.63503397742
-5.00000000000	-5.00000000000	-42.5000000000	2.38506450881	-36.3909573966	9.57226578086
-5.00000000000	-5.00000000000	-42.0000000000	2.47932864781	-37.4124786747	10.4730338056

To switch on the mag field you have to put "yes" to the dipoles in the FootGlobal.par and declare the magnetic map name in TADIdetector.map.

After running makegeo, the input file and the parameters.inc contain all the needed infos to run the simulation with a magnetic field.



Summary of preliminary operations

To be carefully checked before running **makegeo**:

1) Primary type and energy (**TAGdetector.map**)
2) Target material (**TAGdetector.map**)
3) Detectors positions in global coordinates (**FOOT_gep.map**)
4) Detectors activated and FLUKA version (**FootGlobal.par**)

Then in shoe/build/Simulation you can run

../bin/makegeo

to produce foot.inp, foot.geo and parameters.inc.

What MakeGeo actually does

MakeGeo - print of geo: bodies

Printing of the bodies (PrintBodies in TA*base class) for all the detectors and other elements (PrintStandardBodies for blackbody and air, PrintTargBody for target in TAGparGeo class)

MakeGeo.cxx

```
//print bodies
geofile << generalGeo->PrintStandardBodies(  );
geofile << stcGeo->PrintBodies(  );
geofile << bmGeo->PrintBodies(  );
geofile << generalGeo->PrintTargBody(  );
geofile << vtxGeo->PrintBodies(  );
geofile << itrGeo->PrintBodies(  );
geofile << msdGeo->PrintBodies(  );
geofile << diGeo->PrintBodies(  );
geofile << twGeo->PrintBodies(  );
geofile << caGeo->PrintBodies(  );
```

Example for VT: PrintBodies method in TAVTparGeo class

```
if(GlobalPar::GetPar()->IncludeVertex()){

  TAGgeoTrafo* fpFootGeo = (TAGgeoTrafo*)gTAGroot->FindAction(TAGgeoTrafo::GetDefaultActName().Data());

  TVector3  fCenter = fpFootGeo->GetVTCenter();
  TVector3  fAngle = fpFootGeo->GetVTAngles();
  TVector3 posEpi, posPix, posMod;

  string bodyname, regionname;

  ss << "* ***Vertex bodies" << endl;

  for(int iSens=0; iSens<GetNSensors(); iSens++) {

    if(fSensorParameter[iSens].Tilt.Mag()!=0 || fAngle.Mag()!=0)
      ss << "$start_transform " << Form("vt_%d",iSens) << endl;

    //epitaxial layer
    bodyname = Form("vtxe%d",iSens);
    regionname = Form("VTXE%d",iSens);
    posEpi.SetXYZ( fCenter.X() + GetSensorPosition(iSens).X(),
                   fCenter.Y() + GetSensorPosition(iSens).Y(),
                   fCenter.Z() + GetSensorPosition(iSens).Z() - fTotalSize.Z()/2. + fPixThickness + fEpiSize.Z()/2. );
    ss << "RPP " << bodyname << "     "
       << posEpi.x() - fEpiSize.X()/2. << " "
       << posEpi.x() + fEpiSize.X()/2. << " "
       << posEpi.y() - fEpiSize.Y()/2. << " "
       << posEpi.y() + fEpiSize.Y()/2. << " "
       << posEpi.z() - fEpiSize.Z()/2. << " "
       << posEpi.z() + fEpiSize.Z()/2. << endl;
    vEpiBody.push_back(bodyname);
    vEpiRegion.push_back(regionname);

    .....
```

Checks if vertex is included in FootGlobal.par

Retrieves vertex center and rotations in global reference frame

If rotations are present, starts fluka transformation

Calculates position of epitaxial layer

Prints the body for epitaxial layer

Result in foot.geo:

***Vertex bodies

RPP	vtxe0	Xmin: -0.9936 Ymin: -0.96048 Zmin: 0.57814	Xmax: 0.9936 Ymax: 0.96048 Zmax: 0.57954
RPP	vtxm0	Xmin: -0.9948 Ymin: -1.28848 Zmin: 0.5775	Xmax: 1.0292 Ymax: 0.98252 Zmax: 0.5825
RPP	vtxp0	Xmin: -0.9936 Ymin: -0.96048 Zmin: 0.5775	Xmax: 0.9936 Ymax: 0.96048 Zmax: 0.57814
$start_transform		Trans: vt_1	
RPP	vtxe1	Xmin: -0.9936 Ymin: -0.96048 Zmin: 0.89814	Xmax: 0.9936 Ymax: 0.96048 Zmax: 0.89954
RPP	vtxm1	Xmin: -0.9948 Ymin: -1.28848 Zmin: 0.8975	Xmax: 1.0292 Ymax: 0.98252 Zmax: 0.9025
RPP	vtxp1	Xmin: -0.9936 Ymin: -0.96048 Zmin: 0.8975	Xmax: 0.9936 Ymax: 0.96048 Zmax: 0.89814

MakeGeo - print of geo: regions

MakeGeo.cxx

```
//print regions
geofile << generalGeo->PrintStandardRegions1();
geofile << stcGeo->PrintSubtractBodiesFromAir();
geofile << bmGeo->PrintSubtractBodiesFromAir();
geofile << generalGeo->PrintSubtractTargBodyFromAir();
geofile << vtxGeo->PrintSubtractBodiesFromAir();
geofile << itrGeo->PrintSubtractBodiesFromAir();
geofile << msdGeo->PrintSubtractBodiesFromAir();
geofile << diGeo->PrintSubtractBodiesFromAir();
geofile << generalGeo->PrintStandardRegions2();
geofile << twGeo->PrintSubtractBodiesFromAir();
geofile << caGeo->PrintSubtractBodiesFromAir();
geofile << stcGeo->PrintRegions();
geofile << bmGeo->PrintRegions();
geofile << generalGeo->PrintTargRegion();
geofile << vtxGeo->PrintRegions();
geofile << itrGeo->PrintRegio
geofile << msdGeo->PrintRegio
geofile << diGeo->PrintRegion
geofile << twGeo->PrintRegion
geofile << caGeo->PrintRegion
```

Printing of the regions (PrintRegions) for all the detectors and other elements.

```
//___________________________________________________________________
string TAVTparGeo::PrintRegions()
{

  stringstream ss;

  if(GlobalPar::GetPar()->IncludeVertex()){

    string name;

    ss << "* ***Vertex regions" << endl;

    for(int i=0; i<vEpiRegion.size(); i++) {
      ss << setw(13) << setfill( ' ' ) << std::left << vEpiRegion.at(i)
         << "5 " << vEpiBody.at(i) <<endl;
    }

    for(int i=0; i<vModRegion.size(); i++) {
      ss << setw(13) << setfill( ' ' ) << std::left << vModRegion.at(i)
         << "5 " << vModBody.at(i)
         << " -" << vEpiBody.at(i) << " -" << vPixBody.at(i) <<endl;
    }

    for(int i=0; i<vPixRegion.size(); i++) {
      ss << setw(13) << setfill( ' ' ) << std::left << vPixRegion.at(i)
         << "5 " << vPixBody.at(i) <<endl;
    }

  }

  return ss.str();

}
```

Example for VT: PrintRegions method in TAVTparGeo class

```
***Vertex regions
REGION     VTXE0
       expr: vtxe0
REGION     VTXE1
       expr: vtxe1
REGION     VTXE2
       expr: vtxe2
REGION     VTXE3
       expr: vtxe3
REGION     VTXM0
       expr: vtxm0 -vtxe0 -vtxp0
REGION     VTXM1
       expr: vtxm1 -vtxe1 -vtxp1
REGION     VTXM2
       expr: vtxm2 -vtxe2 -vtxp2
```

Result in foot.geo

MakeGeo - print of geo: regions (II)

MakeGeo.cxx

```
//print regions
geofile << generalGeo->PrintStandardRegions1();
geofile << stcGeo->PrintSubtractBodiesFromAir();
geofile << bmGeo->PrintSubtractBodiesFromAir();
geofile << generalGeo->PrintSubtractTargBodyFromAir();
geofile << vtxGeo->PrintSubtractBodiesFromAir();
geofile << itrGeo->PrintSubtractBodiesFromAir();
geofile << msdGeo->PrintSubtractBodiesFromAir();
geofile << diGeo->PrintSubtractBodiesFromAir();
geofile << generalGeo->PrintStandardRegions2();
geofile << twGeo->PrintSubtractBodiesFromAir();
geofile << caGeo->PrintSubtractBodiesFromAir();
geofile << stcGeo->PrintRegions();
geofile << bmGeo->PrintRegions();
geofile << generalGeo->PrintTargRegion();
geofile << vtxGeo->PrintRegions();
geofile << itrGeo->PrintRegions();
geofile << msdGeo->PrintRegions(  );
geofile << diGeo->PrintRegions(  );
geofile << twGeo->PrintRegions(  );
geofile << caGeo->PrintRegions(  );
```

All the detectors are subtracted from air (PrintSubtractBodiesFromAir)

Example for VT:
PrintSubtractBodiesFromAir method in TAVTparGeo class

```
//_____________________________________________
string TAVTparGeo::PrintSubtractBodiesFromAir()
{

  stringstream ss;

  if(GlobalPar::GetPar()->IncludeVertex()){

    for(int i=0; i<vModBody.size(); i++) {
      ss << " -" << vModBody.at(i);
    }
    ss << endl;

  }

   return ss.str();
}
```

Result in foot.geo

```
***Air
REGION          AIR1
           expr: air +airpla-stc
                 -(BmnShiOu -BmnShiIn) -(BmnShiIn -BmnMyl0 +BmnMyl3)
                 -vtxm0 -vtxm1 -vtxm2 -vtxm3
```

MakeGeo - print of input: beam

MakeGeo.cxx

```
outfile << generalGeo->PrintBeam();
outfile << generalGeo->PrintPhysics();
```

```
stringstream str;

string part_type;
if (GetBeamPar().AtomicNumber>2)
  part_type = "HEAVYION";
else if (GetBeamPar().AtomicNumber==1 && GetBeamPar().AtomicMass==1)
  part_type = "PROTON";
else if (GetBeamPar().AtomicNumber==2 && GetBeamPar().AtomicMass==4)
  part_type = "4-HELIUM";
else{
  cout << "**** ATTENTION: unknown beam!!!! ****"<< endl;
  exit(0);
}

str << PrintCard("BEAM",TString::Format("%f",-(GetBeamPar().Energy)), "",
                 TString::Format("%f",GetBeamPar().AngDiv),
                 TString::Format("%f",-GetBeamPar().Size),
                 TString::Format("%f",-GetBeamPar().Size),
                 "1.0",part_type) << endl;
if(part_type == "HEAVYION")
  str << PrintCard("HI-PROPE",TString::Format("%d",GetBeamPar().AtomicNumber),
                   TString::Format("%.0f",GetBeamPar().AtomicMass),"","","","","") << endl;
str << PrintCard("BEAMPOS",TString::Format("%.3f",GetBeamPar().Position.X()),
                 TString::Format("%.3f",GetBeamPar().Position.Y()),
                 TString::Format("%.3f",GetBeamPar().Position.Z()),"","","","") << endl;

return str.str();
```

PrintBeam is a method of TAGparGeo class (shoe/libs/src/TAMCbase/TAGparGeo.cxx)
It gets the parameters (beam A, Z, pos, ...) from shoe/build/Reconstruction/level0/geomaps/TAGparGeo.map

Result in foot.inp

BEAM	Beam: Energy ▾	E: 0.2	Part: HEAVYION ▾
Δp: Flat ▾	Δp:	Δϕ: Flat ▾	Δϕ: 0
Shape(X): Gauss ▾	x(FWHM): 0.48	Shape(Y): Gauss ▾	y(FWHM): 0.48
HI-PROPE	Z: 6	A: 12	Isom:
BEAMPOS	x: 0	y: 0	z: -30
	cosx:	cosy:	Type: POSITIVE ▾

MakeGeo - print of input: physics

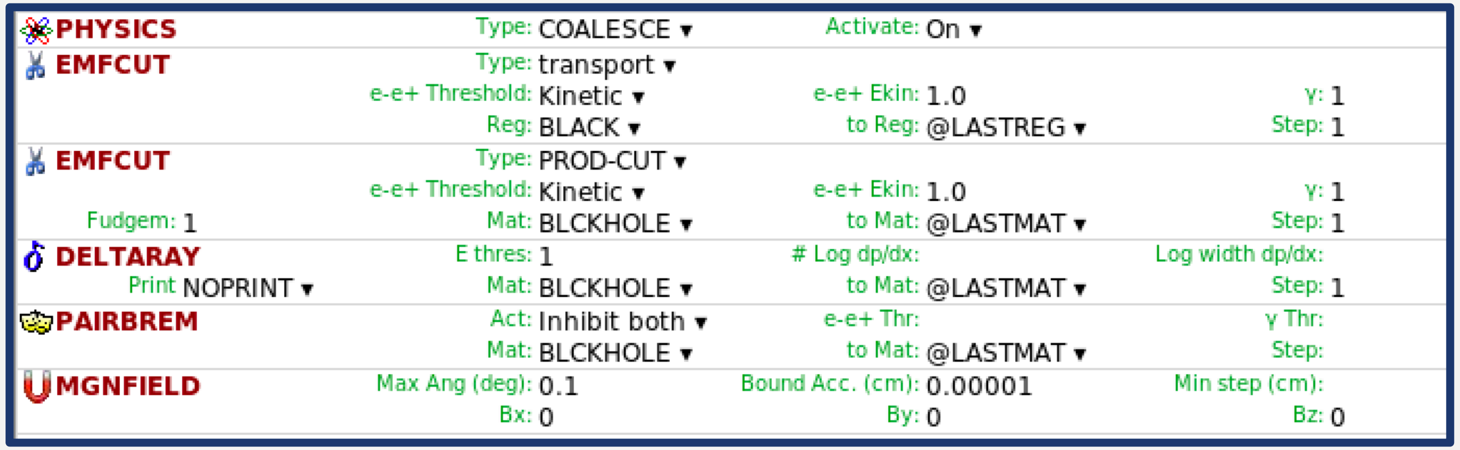
MakeGeo.cxx

```
outfile << generalGeo->PrintBeam();
outfile << generalGeo->PrintPhysics();
```

```
stringstream str;

if ( GlobalPar::GetPar()->verFLUKA() )
  str << PrintCard("PHYSICS","1.","","","","","","COALESCE") << endl;
else
  str << PrintCard("PHYSICS","12001.","1.","1.","","","","COALESCE") << endl;

str << PrintCard("EMFCUT","-1.","1.","","BLACK","@LASTREG","1.0","") << endl;
str << PrintCard("EMFCUT","-1.","1.","1.","BLCKHOLE","@LASTMAT","1.0","PROD-CUT") << endl;
str << PrintCard("DELTARAY","1.","","","BLCKHOLE","@LASTMAT","1.0","") << endl;
str << PrintCard("PAIRBREM","-3.","","","BLCKHOLE","@LASTMAT","","") << endl;

if(GlobalPar::GetPar()->IncludeDI()){
  str << PrintCard("MGNFIELD","0.1","0.00001","","0.","0.","0.","") << endl;
}

return str.str();
```

PrintPhysics is a method of TAGparGeo class (shoe/libs/src/TAMCbase/TAGparGeo.cxx). It is, at present, hardcoded → to be changed. It handles transport threshold, as well as the magnetic field. If the dipoles are included in FootGlobal.par, the card MGNFIELD is printed. It calls the routine magfld.f that handles the magnetic field (more details in the following).

Result in foot.inp



MakeGeo - print of input: materials (I)

MakeGeo.cxx

```
//print materials and compounds
outfile << fTAGmat->PrintMaterialFluka();

//print assig nmaterials
outfile << generalGeo->PrintStandardAssignMaterial();
outfile << stcGeo->PrintAssignMaterial(fTAGmat);
outfile << bmGeo->PrintAssignMaterial(fTAGmat);
outfile << generalGeo->PrintTargAssignMaterial(fTAGmat);
outfile << vtxGeo->PrintAssignMaterial(fTAGmat);
outfile << itrGeo->PrintAssignMaterial(fTAGmat);
outfile << msdGeo->PrintAssignMaterial(fTAGmat);
outfile << diGeo->PrintAssignMaterial(fTAGmat);
outfile << twGeo->PrintAssignMaterial(fTAGmat);
outfile << caGeo->PrintAssignMaterial(fTAGmat);
```

Method of TAGmaterials class, which writes the cards needed to define the materials (MATERIAL and COMPOUND). It integrates the FLUKA standard materials and the materials defined in shoe.

Example of materials in foot.inp

MATERIAL	Name: BISMUTH	#	ρ: 9.747
Z: 83	Am: 208.98	A:	dE/dx: ▾
MATERIAL	Name: GERMANIU	#	ρ: 5.323
Z: 32	Am: 72.61	A:	dE/dx: ▾
MATERIAL	Name: BGO	#	ρ: 7.13
Z:	Am:	A:	dE/dx: ▾

MakeGeo - print of input: materials (II)

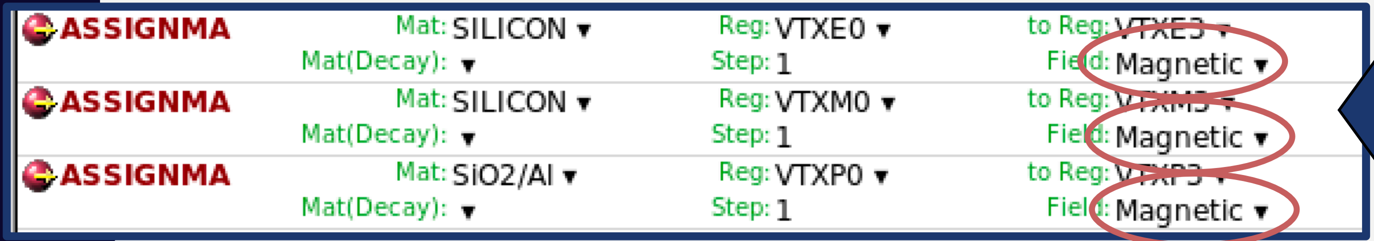
MakeGeo.cxx

```
//print materials and compounds
outfile << fTAGmat->PrintMaterialFluka();

//print assig nmaterials
outfile << generalGeo->PrintStandardAssignMaterial();
outfile << stcGeo->PrintAssignMaterial(fTAGmat);
outfile << bmGeo->PrintAssignMaterial(fTAGmat);
outfile << generalGeo->PrintTargAssignMaterial(fTAGmat);
outfile << vtxGeo->PrintAssignMaterial(fTAGmat);
outfile << itrGeo->PrintAssignMaterial(fTAGmat);
outfile << msdGeo->PrintAssignMaterial(fTAGmat);
outfile << diGeo->PrintAssignMaterial(fTAGmat);
outfile << twGeo->PrintAssignMaterial(fTAGmat);
outfile << caGeo->PrintAssignMaterial(fTAGmat);
```

Example for VT: PrintAssignMaterial method in TAVTparGeo class

```
//______________________________________________________________________
string TAVTparGeo::PrintAssignMaterial(TAGmaterials *Material)
{

  stringstream ss;

  if(GlobalPar::GetPar()->IncludeVertex()){

    TString flkmatMod, flkmatPix;

    if (Material == NULL){
      TAGmaterials::Instance()->PrintMaterialFluka();
      flkmatMod = TAGmaterials::Instance()->GetFlukaMatName(fEpiMat.Data());
      flkmatPix = TAGmaterials::Instance()->GetFlukaMatName(fPixMat.Data());
    }else{
      flkmatMod = Material->GetFlukaMatName(fEpiMat.Data());
      flkmatPix = Material->GetFlukaMatName(fPixMat.Data());
    }

    bool magnetic = false;
    if(GlobalPar::GetPar()->IncludeDI())
      magnetic = true;

    if (vEpiRegion.size()==0 || vModRegion.size()==0 || vPixRegion.size()==0 )
      cout << "Error: VT regions vector not correctly filled!"<<endl;

    ss << PrintCard("ASSIGNMA", flkmatMod, vEpiRegion.at(0), vEpiRegion.back(),
                    "1.", Form("%d",magnetic), "", "") << endl;
    ss << PrintCard("ASSIGNMA", flkmatMod, vModRegion.at(0), vModRegion.back(),
                    "1.", Form("%d",magnetic), "", "") << endl;
    ss << PrintCard("ASSIGNMA", flkmatPix, vPixRegion.at(0), vPixRegion.back(),
                    "1.", Form("%d",magnetic), "", "") << endl;

  }

  return ss.str();

}
```

Result in foot.inp



The PrintAssignMaterial method, beside associate the region with their material, checks if the magnetic field is present and activates (or not) in the card ASSIGNMA the switch Magnetic, so that the simulation considers the magnetic field inside that region.

MakeGeo - print of input: rotations (I)

MakeGeo.cxx

```
// print rotations
outfile << stcGeo->PrintRotations();
outfile << bmGeo->PrintRotations();
outfile << generalGeo->PrintTargRotations();
outfile << vtxGeo->PrintRotations();
outfile << itrGeo->PrintRotations();
outfile << msdGeo->PrintRotations();
outfile << diGeo->PrintRotations();
outfile << twGeo->PrintRotations();
outfile << caGeo->PrintRotations();
```

```
if(GlobalPar::GetPar()->IncludeVertex()){

  TAGgeoTrafo* fpFootGeo = (TAGgeoTrafo*)gTAGroot->FindAction(TAGgeoTrafo::GetDefaultActName().Data());

  TVector3  fCenter = fpFootGeo->GetVTCenter();
  TVector3  fAngle = fpFootGeo->GetVTAngles();

  for(int iSens=0; iSens<GetNSensors(); iSens++) {

    //check if sensor or detector have a tilt
    if (fSensorParameter[iSens].Tilt.Mag()!=0 || fAngle.Mag()!=0){

      //put the sensor in local coord before the rotation
      ss << PrintCard("ROT-DEFI", "", "", "",
                      Form("%f",-fCenter.X()),
                      Form("%f",-fCenter.Y()),
                      Form("%f",-fCenter.Z()),
                      Form("vt_%d",iSens) ) << endl;

      //check if sensor has a tilt
      if (fSensorParameter[iSens].Tilt.Mag()!=0){

        // put the sensor in 0,0,0 before the sensor's rot
        ss << PrintCard("ROT-DEFI", "", "", "",
                        Form("%f",-GetSensorPosition(iSens).X()),
                        Form("%f",-GetSensorPosition(iSens).Y()),
                        Form("%f",-GetSensorPosition(iSens).Z()),
                        Form("vt_%d",iSens) ) << endl;
        //rot around x
        if(fSensorParameter[iSens].Tilt[0]!=0){
          ss << PrintCard("ROT-DEFI", "100.", "",
                          Form("%f",fSensorParameter[iSens].Tilt[0]*TMath::RadToDeg()),
                          "", "", "", Form("vt_%d",iSens) ) << endl;
        }
        //......same for y and z
```

```
        //put back the sensor into its position in local coord
        ss << PrintCard("ROT-DEFI", "", "", "",
                        Form("%f",GetSensorPosition(iSens).X()),
                        Form("%f",GetSensorPosition(iSens).Y()),
                        Form("%f",GetSensorPosition(iSens).Z()),
                        Form("vt_%d",iSens) ) << endl;
      }

      //check if detector has a tilt and then apply rot
      if(fAngle.Mag()!=0){

        if(fAngle.X()!=0){
          ss << PrintCard("ROT-DEFI", "100.", "", Form("%f",fAngle.X()),"", "",
                          "", Form("vt_%d",iSens)) << endl;
        }
        //......same for y and z
      }

      //put back the detector in global coord
      ss << PrintCard("ROT-DEFI", "", "", "",
                      Form("%f",fCenter.X()), Form("%f",fCenter.Y()),
                      Form("%f",fCenter.Z()), Form("vt_%d",iSens)) << endl;
    }
```

Card	Axis	Polar	Δx	Id	Azm	Δy	Name	Δz
ROT-DEFI	Z		0	0		0	vt_1	-1.500000
ROT-DEFI	Z		0	0		0	vt_1	0.600000
ROT-DEFI	Y			0	180		vt_1	
ROT-DEFI	Z		0	0		0	vt_1	-0.600000
ROT-DEFI	Z		0	0		0	vt_1	1.500000
ROT-DEFI	Z		0	0		0	vt_3	-1.500000
ROT-DEFI	Z		0	0		0	vt_3	-0.920000
ROT-DEFI	Y			0	180		vt_3	
ROT-DEFI	Z		0	0		0	vt_3	0.920000
ROT-DEFI	Z		0	0		0	vt_3	1.500000

Result in foot.inp

In FLUKA it's possible to perform only rotations around the 3 axes of the reference frame. So, to rotate for example a VT sensor around its own axis one has firstly to shift the sensor in (0, 0, 0), then rotate it, and finally put the sensor back in its global position.

MakeGeo - print of input: rotations (II)

In SHOE two "types" of rotations are defined:

- Rotations of the whole detector, reported → FOOT_geo.map
- Rotations of single parts of the detector (ex. single VT sensors) for the detectors which are "segmented" → TA*detector.map

The rotations implemented are:

Detector	Whole detector			Single Sensor		
	X	Y	Z	X	Y	Z
ST	✔	✔	✔			
BM	✔	✔	✔			
TG	✔	✔	✔			
VT	✔	✔	✔	✔	✔	✔
IT	✔	✔	✔	✔	✔	✔
MSD	✔	✔	✔	✔	✔	✔
DI	✔	✔	✔	✔	✔	✔
TW	✔	✔	✔	✔	✔	✔
CA	✔	✔	✔	✔	✔	✔

MakeGeo - print of parameters.inc

MakeGeo.cxx

```
//parameter file needed by the user routines
parFileName = Form("./ROUTINES/parameters.inc");
ofstream paramfile;
paramfile.open(parFileName);
if ( !paramfile.is_open() )
  cout<< "ERROR  --> I do not find the parameters.inc file"<<fileName.c_str()<< endl;

paramfile << bmGeo->PrintParameters();
paramfile << vtxGeo->PrintParameters();
paramfile << itrGeo->PrintParameters();
paramfile << msdGeo->PrintParameters();
paramfile << diGeo->PrintParameters();
paramfile << twGeo->PrintParameters();
paramfile << caGeo->PrintParameters();

paramfile.close();
```

```
//________________________________________________________________________
string TAVTparGeo::PrintParameters()
{
  stringstream outstr;

  if(GlobalPar::GetPar()->IncludeVertex()){

    string precision = "D+00";

    outstr << "c     VERTEX PARAMETERS " << endl;
    outstr << endl;

    map<string, int> intp;
    intp["nlayVTX"] = fSensorsN;
    for (auto i : intp){
      outstr << "      integer " << i.first << endl;
      outstr << "      parameter (" << i.first << " = " << i.second << ")" << endl;
    }

    outstr << endl;
  }

  return outstr.str();
}
```

parameters.inc is an include file needed by the user routines. It stores infos useful to run quite automatedly the simulation and it is written in FORTRAN.

Result in parameters.inc

```
c     VERTEX PARAMETERS

      integer nlayVTX
      parameter (nlayVTX = 4)
```

Building the FLUKA executable

Scripts to link and compile the routines

The user routines, together with the parameter.inc file created by makegeo, must be compiled and linked to produce the correct executable to run the simulation. This executable will assure the creation of the FOOT customized output (see FOOT user routines presentation).

Therefore, the user must run one of the following according whether the magnetic field must be simulated or not:

- *source link_FOOT.sh* → NO magnetic field
- *source link_FOOT_mag.sh* → magnetic field

This will respectively produce an executable:

- fluka_FOOT.exe
- fluka_FOOT_mag.exe

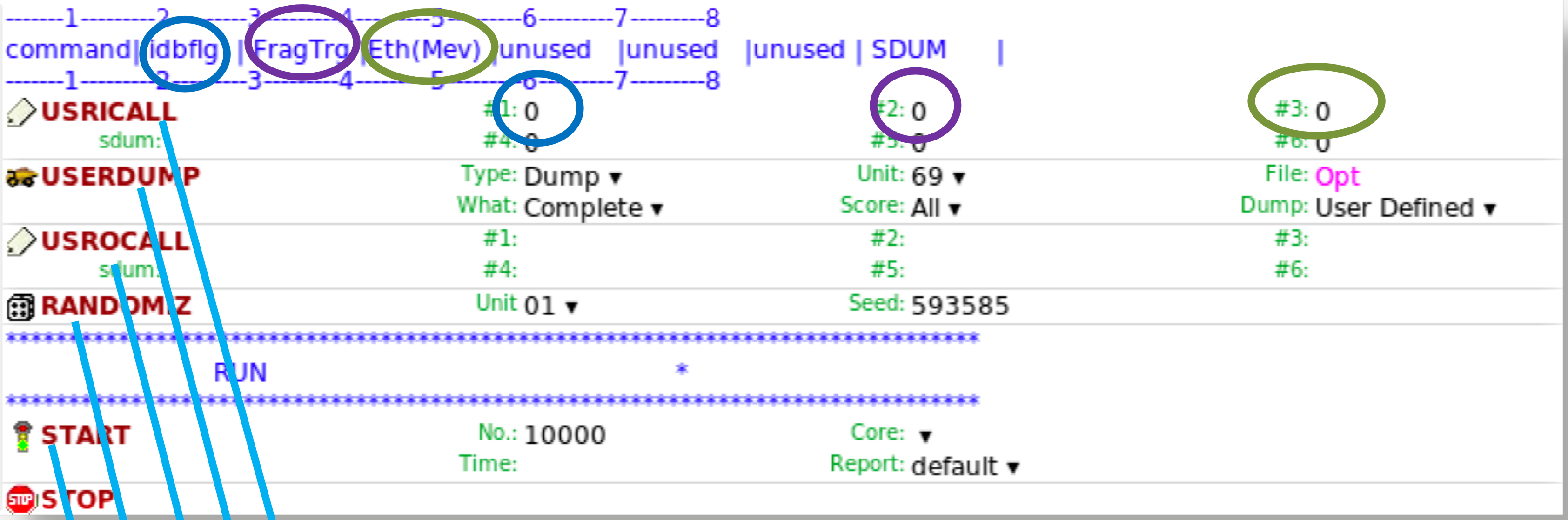
that must be then used when launching the simulation.

There are also the scripts for the FLUKA development version (reserved to developers).

Running the simulation

Before the simulation (I)

Notice that not all the cards in the input file are modified by makegeo:



Further information in the slides about user routines.

It calls the user-written routine usrini.f. The user can set in particular a debug flag (if >0 a verbose event debug output is written on the *.log file) and a trigger. The last one specifies what kind of event will be recorded in the output (i.e. FragTrg=6 to register only events with target fragmentation, FragTrg=0 to register all events).

Calls the routine usrout.f. Do not modify this card.

This command activates calls to the user routine mgraw.f. Do not modify it.

Sets the random seed number.

Sets the number of primary. You can modify it by hand according to your purpose.

Before the simulation (II)

To be carefully checked :

1) Trigger flag to write events in the USRICALL card ❋
2) Number of primaries in the START card

The foot.inp file may of course be renamed to any useful <name>.inp

❋ Untriggered output means that ~98.5% of events are primaries not interacting in the target. They might interact in VT, IT, MSD or TW and eventually die in CA, producing there many particles ➞ Very large TXT file!

Running the job

The executable thus produced (*fluka_FOOT.exe* or *fluka_FOOT_mag.exe*) has to be run with the proper FLUKA script to launcg the simulation:

$FLUPRO/flutil/rfluka –e fluka_FOOT.exe –N0 –M4*

foot This will launch a run with 4 cycles for the input file foot.inp.

Depending on the number of available core it is possible to send several runs (each one for a different input file!!) in parallel.

For each case a temporary fluka_xxxx directory will be created.

Progress of job can be checked looking at the tail of *.out or *.err file in the temporary directory.

Example: 380000 620000 620000 1.0134497E-02 1.0000000E+30

- 380000: No. of events processed
- 620000: No. of events still to be processed
- 1.0134497E-02: Average cpu time/event at that time

- To stop a cycle : in fluka_xxxx create fluka.stop (*touch fluka.stop*)
- To stop a run: *touch rfluka.stop*

At the end of the job

As a first thing, check in the *.log files possible messages of error.

Among other files produced by running the simulation, in each cycle a *TXT.dat file is created.

Example:

```
ls -1 *TXT.dat
```

```
>   foot001_TXT.dat        foot003_TXT.dat
    foot002_TXT.dat        foot004_TXT.dat
```

The command

```
ls -1 *TXT.dat > foot.lis
```

will create a foot.lis file containing the list of files to be processed to create the root output file.

After the simulation run

Converting the output into a ROOT file

The TXT files will be converted to a ROOT tree by means of a software: Txt2Root. This is automatically compiled when you compile the code in shoe/build/Simulation, and can be found in shoe/build/bin.

To convert a single .dat file:

../bin/Txt2Root –in foot001_TXT.dat -out outputname.root

To convert a list of .dat files:

../bin/Txt2Root –in foot.lis -iL -out outputname.root

This switch must be used to say that <name>.lis is a list of files

The resulting ROOT file has the usual ntuple structure (see FOOT output presentation) and it can be processed by the shoe reconstruction code.

Thank you ☺