GEMMA1.6 Geant4 Simulation Documentation

Naomi Galinski

April 2014

Contents

Note: Throughout this document NuTemma refers to and is linked to the online Numerical Tools for EMMA website `http://davids24.triumf.ca/~oliver/NUTEMMA/home.html`.

1 Installations

1.1 Geant4 Installation

There are instructions in the Download tab of the GEMMA section in NuTemma on how to install the necessary programs to run GEMMA. I am running Mac OS X 10.8 and the following instructions worked for my computer.

1. First install **cmake** which is used to build and install Geant4 on your computer. This generates the make file with the specified compiler options. I installed it with MacPorts using the command:

   sudo port install cmake

2. You also need to install **f2c** the Fortran-to-C converter. You can do this by typing:

   sudo port install f2c

3. I have **geant4.9.6.p02** installed. Download the source file and data files from Geant4 Software Download following the instructions in the Geant4 Installation Guide. I highly recommend you read it to understand the Geant4 installation process. I chose to install Geant4 in a separate build directory from the source folder, but you don't have to do that. Here are my detailed instructions of what I did:

   i Unpack and copy Geant4 folder to **/usr/local/geant/**

   ii Unpack the data files and copy them into **/usr/local/geant/geant4.9.6.p02/share/Geant4-9.6.2/data**

   iii **cd /usr/local/geant/**

   iv **mkdir /usr/local/geant/geant4.9.6.p02-build**

   v **cd /usr/local/geant/geant4.9.6.p02-build**

vi sudo cmake -DGeant4_USE_OPENGL_X11=ON -DGeant4_USE_RAYTRACER_X11=ON -DGeant4_INSTALL_DATADIR=/usr/local/geant/geant4.9.6.p02/share/Geant4-9.6.2/data -DGeant4_INSTALL_DATA=ON -DCMAKE_INSTALL_PREFIX=/usr/local/geant/geant4.9.6.p02 /usr/local/geant/geant4.9.6.p02

The options Geant4_INSTALL_DATA, Geant4_USE_OPENGL_X11 and Geant4_USE_RAYTRACER_X11 are important here. The options DGeant4_INSTALL_DATADIR=/usr/local/geant/geant4.9.6.p02/share/Geant4-9.6.2/data and -DGeant4_INSTALL_DATA=ON are needed to install the data packages.

vii sudo make -j2

If you have a two core processor. You can leave out the -jncore argument if you want.

viii sudo make install

4. To link to the Geant4 libraries so that you can run it from any terminal put the following five lines into your .bash_profile

```
alias cmake-geant4='cmake -DGeant4_DIR=/usr/local/geant/geant4.9.6.p02'
export PATH=$PATH:/usr/local/geant/geant4.9.6.p02/bin
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/geant/geant4.9.6.p02/bin
export DYLD_LIBRARY_PATH=$DYLD_LIBRARY_PATH:$LD_LIBRARY_PATH
. /usr/local/geant/geant4.9.6.p02/bin/geant4.sh
```

1.2 ROOT Installation

I have ROOT version 5.34.00 running on my Mac. I tried installing ROOT from source following the download and installation instructions, however I didn't succeed. Instead I downloaded a binary version, from the same page. I feel lucky that a precompiled version worked. I moved this folder to:

/usr/local/cern-root/5.34.00

To link to the ROOT libraries correctly add the following to your .bash_profile

export ROOTSYS=/usr/local/cern-root/5.34.00

export PATH=$PATH:$ROOTSYS/bin

export MANPATH=$MANPATH:$ROOTSYS/man

If you can't get ROOT to run GEMMA will still run. There are ROOT flags in the simulation such that if the compiler doesn't find the ROOT libraries it ignores everything between the flags.

2 Compiling GEMMA

1. First download and acquire gemma1.6.zip (not available online at time I'm writing this).
2. Make sure that the folder and files contained within are read and writable. If not then compiling using **make** will fail. If the files are not read and writable or you're not sure then you can type the following command to give read and write permission to all folders and files within:

   chmod -R ugo+rw gemma1.6
3. **CMakeLists.txt** contains all the make options.

```
#----------------------------------------------------------------------------
# Setup the project
cmake_minimum_required(VERSION 2.6 FATAL_ERROR)
project(EMMA)

#----------------------------------------------------------------------------
# Find Geant4 package, activating all available UI and Vis drivers by default
# You can set WITH_GEANT4_UIVIS to OFF via the command line or ccmake/cmake-gui
# to build a batch mode only executable
#
option(WITH_GEANT4_UIVIS "Build example with Geant4 UI and Vis drivers" ON)
if(WITH_GEANT4_UIVIS)
  find_package(Geant4 REQUIRED ui_all vis_all)
else()
  find_package(Geant4 REQUIRED)
endif()

#----------------------------------------------------------------------------
```

```
# Setup Geant4 include directories and compile definitions
# Setup include directory for this project
#
include(${Geant4_USE_FILE})

#----------------------------------------------------------------------------
# Find ROOT (optional package)
#
find_package(ROOT)
if(ROOT_FOUND)
  add_definitions(-DG4ANALYSIS_USE)
endif()

#----------------------------------------------------------------------------
# Locate sources and headers for this project
# NB: headers are included so they will show up in IDEs
#
include_directories(${CMAKE_CURRENT_SOURCE_DIR}/include
                    ${Geant4_INCLUDE_DIR}
                    ${ROOT_INCLUDE_DIR})
file(GLOB sources ${PROJECT_SOURCE_DIR}/src/*.cc)
file(GLOB headers ${PROJECT_SOURCE_DIR}/include/*.hh)

#----------------------------------------------------------------------------
# Add the executable, and link it to the Geant4 libraries
#
set(F2C_LIBRARIES /opt/local/lib/libf2c.a)
add_executable(EMMAapp EMMAapp.cc ${sources} ${headers})
target_link_libraries(EMMAapp ${Geant4_LIBRARIES} ${ROOT_LIBRARIES}
${F2C_LIBRARIES} )

#----------------------------------------------------------------------------
# Copy all scripts to the build directory, i.e. the directory in which we
# build A01. This is so that we can run the executable directly because it
# relies on these scripts being in the current working directory.
#
set(EMMA_SCRIPTS
   BeamSetup.mac visEMMA.mac
  )

foreach(_script ${EMMA_SCRIPTS})
```

```
  configure_file(
    ${PROJECT_SOURCE_DIR}/${_script}
    ${PROJECT_BINARY_DIR}/${_script}
    COPYONLY
    )
endforeach()

#----------------------------------------------------------------------------
# Add program to the project targets
# (this avoids the need of typing the program name after make)
#
add_custom_target(EMMA DEPENDS EMMAapp)

#----------------------------------------------------------------------------
# Install the executable to 'bin' directory under CMAKE_INSTALL_PREFIX
#
install(TARGETS EMMAapp DESTINATION bin)
```

You don't need to make any changes, but note the following: visualization is turned ON, if ROOT is found it will compile everything between the **G4ANALYSIS_USE** flags in the code, and **F2C** libraries are included as well as all header and source files in the **/include/** and **/src/** folders respectively.

4. Next go into the **cmake_script.sh** shell script and change the last line to your build directory. The simulation requires the folders **fortran**, and **UserDir**, therefore I suggest using the same **/gemma1.6/** directory as the build directory. If you chose a separate build directory such as in the case of the Geant4 installation remember to copy the two folders into the build directory. The build directory doesn't contain the cmake files and is therefore less cluttered, that's all.

```
rm Makefile
rm CMakeCache.txt
rm -r CMakeFiles
rm cmake_install.cmake
cmake -DGeant4_DIR=/usr/local/geant/geant4.9.6.p02 /Users/naomig/Documents/EMMA/gemma1.5
```

Change the last argument in the last line to where you put **gemma1.6**

5. To compile **gemma1.6** run

./cmakescript.sh
make

You only ever need to run cmake again if you've added/removed new source (.cc) or header (.hh) files. Otherwise compile with **make** only each time you make a changes to files.

3 Running GEMMA

If the compilation was successful, you can now run GEMMA by typing:

./EMMAapp

You will be using the command line to start simulations, visualize the geometry, track particles and make changes to input parameters. I will list some of the most used commands:

1. **exit** Exit the program. Note: If you are using ROOT to analyze the data then you must exit the program before you can open the file with ROOT.
2. **/control/execute visEMMA.mac** Open X11 window, display EMMA and particle tracks.
3. **/mydet/doBeam** Simulate beam particles through EMMA.
4. **/mydet/doPrepare** Calculate the energy, position and direction of the beam particles at random reaction depths.
5. **/mydet/doReaction** Calculate the recoil energy and direction at the reaction depth from the energy and direction of the beam particle calculated with **/mydet/doPrepare** and transfer reaction kinematics. To simulate recoils you must execute **/mydet/doPrepare** first.
6. **/mydet/nEvents n** Change number of events. Substitute **n** for the number of particles you desire.

I have turned off the auto run setting. If you want to automatically start simulations on startup of the program then uncomment the code around line 220 in **EMMAapp.cc** and modify as needed.

To view all command line directories type **ls** in the terminal. You can also type **ls /directoryname/** to look at a list of sub-directories and commands within. Type **help commandname** to get more information about the command.

4 Visualization

visEMMA.mac is a macro that when executed will display EMMA's elements and show particle tracks. The elements are called logical volumes in Geant4. **visdebug.mac** is a macro I used to display logical volumes only. I used this as a debugger for positioning of the volumes. Both files are executed with **/control/execute filename**. Check out both files for examples of Geant4 visualization commands.

Here is a list of references I found useful:

- Geant4 Visualization Tutorial using the OpenGL Event Display
- 8.4. Controlling Visualization from Commands
- List of visualization commands

5 Input Files

There are seven input files that are read into the simulation. The following lists the file names and example contents. Most of the parameters are explained in the Manual tab of the GEMMA section of NuTemma.

1. **/UserDir/UserInput/beam.dat** (opened in EMMAapp.cc)

```
100     # number of events
50      # beam Z
132     # beam A
30      # beam charge state Q
264     # beam kinetic energy (MeV)
0.0     # fractional energy spread (FWHM)
1.0     # beam-spot diameter (mm)
0.0     # normalized transverse emittance (pi mm mrad)
```

   This file contains the number of events to be simulated and the beam properties.

2. **/UserDir/UserInput/reaction.dat** (opened in EMMAapp.cc)

```
# Two-body reaction: 1+2->3+4
50       # Z1 beam
132      # A1
```

```
1        # Z2 target
2        # A2
50       # Z3 recoil
133      # A3
1        # Z4 ejectile
1        # A4
157.3    # min c.m. angle of ejectile
180.0    # max c.m. angle of ejectile
30       # charge state of recoil
0.0      # excitation energy of recoil
7        # solid-angle averaged cross section (mb/sr) inside the angular range specified above
```

The charge and mass numbers of the reaction particles are listed first. The recoil angles were calculated using the Relativistic Two-Body Kinematics Calculator and Plotter. The charge state of the recoil can be obtained using the charge state calculator in NuTemma. The cross section, the last input parameter, IS NOT USED in version gemma1.6.

3. **/UserDir/UserInput/centralTrajectory.dat** (opened in EMMAapp.cc)

```
50       # Z
133      # A
30       # charge state Q
258      # kinetic energy (MeV)
```

Central recoil properties. The energy loss through and the charge state after the target (and the degrader if used) were obtained using the energy loss and charge state calculator in NuTemma.

4. **/UserDir/UserInput/targetDegraders.dat** (opened in EMMADetectorConstruction.cc)

```
# TARGET
IN                 # IN/OUT
0.001              # thickness (um)
20                 # distance to Q1 (cm)
1.06               # density (g/cm3)
2                  # number of elements
1 2.014 0.25       # element 1: Z, M (g/mol), mass fraction
6 12.01 0.75       # element 2: Z, M (g/mol), mass fraction
# DEGRADER 1
OUT                # IN/OUT
9                  # thickness (um)
```

```
1                   # distance from target (cm)
19.30               # density (g/cm3)
1                   # number of elements
79 197.0 1          # element 1: Z, M (g/mol), weight ratio
# DEGRADER 2
OUT
0
0
0
0
# DISPLACE DETECTOR
0
```

Target and degrader specifications. Up to two degraders can be used to slow down the recoils.

5. **/UserDir/UserInput/mwpc.dat** (opened in EMMADetectorConstruction.cc)

```
OUT       # IN/OUT
0         # pressure (torr)
0         # temperature (degrees Celsius)
```

Multiwire Proportional Counter setup specifications. The MPCW is placed at the focal plane and consists of a gas chamber which is 75 mm in length and filled with isobutane.

6. **/UserDir/UserInput/slits.dat** (opened in SpectrometerConstruction.cc)

```
# SLITS 1
# HORIZONTAL
OUT                 # IN/OUT
0                   # aperture (mm)
# SLITS 2
# HORIZONTAL
OUT                 # IN/OUT
0                   # aperture (mm)
# SLITS 3
# HORIZONTAL
OUT                 # IN/OUT
0                   # aperture (mm)
# VERTICAL
OUT                 # IN/OUT
0                   # aperture (mm)
```

Vertical and horizontal slit placements. Slit 1 is placed in the drift space between ED1

and MD, slit 2 between MD and ED2 and slit 3 between Q4 and the focal plane. These are used to limit the spread of beam particles and recoils.

7. **/UserDir/UserInput/alphaSource.dat** (opened in EMMAPrimaryGeneratorAction.cc)

```
NO      # YES/NO
5.5     # alpha particle kinetic energy (MeV)
2.0     # max angle alpha source (deg)
```

Alpha source parameters. Alpha particles can be simulated going through EMMA instead of a beam.

6 Output Files

The two important files you are interested in are

- **/UserDir/Results/fp_beam.dat** and
- **/UserDir/Results/fp_reaction.dat**.

For one event the contents of **/UserDir/Results/fp_beam.dat** is

```
319.379, 2.60606, 7.52608, 2.47199
```

which are the $kinetic\,energy\,(MeV)$, $angle\,(deg)$, $x\,position\,(cm)$, $y\,position\,(cm)$ of the beam particle that hit the focal plane. The files are opened in **EMMAPrimaryGeneratorAction.cc** and written to in **EMMADriftChamberHit.cc** which is in turn called by **EMMAEventAction.cc**. More on these classes will be discussed in the next section.

When you run the simulation to prepare the beam for the reaction kinematics with the **/mydet/doPrepare** command the

- **/UserDir/BeamSampling/beam.dat**

file is created. It contains the kinetic energy, position and momentum of the beam particle at the reaction dept. It is opened in **EMMAPrimaryGeneratorAction.cc** and the data is written to in **EMMASteppingAction.cc**. This file is opened again and its contents read in when simulating recoils with the **/mydet/doReaction** command. It is used to calculate the recoil energy and direction using transfer reaction kinematics.

The files

- **/userDir/Results/postTarget_beam.dat**
- **/userDir/Results/postDegrader1_beam.dat**
- **/userDir/Results/postTarget_reaction.dat**
- **/userDir/Results/postDegrader1_reaction.dat**

contain energies and angles of the beam particles or recoils just after the target and degrader. The comments say that it is needed to optimize the electric and magnetic field strengths, but the outputs of these files are not used elsewhere in the code. Instead the **/UserDir/UserInput/centralTrajectory.dat** data is used to calculate the optimal field strengths.

The ROOT file

- **/userDir/Results/GEMMAoutput.root**

is created in **EMMAEventAction.cc** if ROOT is linked properly. It contains histograms and a tree of the positions and angles of the particles at the focal plane. These are written into the file in **EMMAEventAction.cc**. The location of dead hits for those particles that don't make it to the focal place are stored in another histogram. This is written to file in **EMMASteppingAction.cc**. You can add the data from both beam and recoil in the same ROOT file by running **/mydet/doBeam**, **/mydet/doPrepare** and **/mydet/doReaction** in the same session. Don't forget the **exit** the program before you proceed to look at the file in ROOT.

A ROOT macro called **/userDir/Results/rootanalysis/GEMMArootanalysis.C** was written to automatically read in the file, plot the histograms and read in the event by event information from a tree. The macro can be run in ROOT as following:

1. **cd userDir/Results/rootanalysis/**
2. **root**
3. **.x GEMMArootanalysis.C+**

7 How the program works

I suggest you read the following manuals in parallel to trying to understand how Geant4 and the program works:

- Geant4 User's Guide: This is a shorter manual.
- Geant4 User's Guide for Application Developers: This is an extensive manual.
- Geant4 Cross Reference: This is the most useful in searching for Geant4 files, classes, methods, variables, etc. It searches through all Geant4 files including example simulations.

7.1 Main Program

EMMAapp.cc is the main program. Here we define all user initialization and action classes using **G4RunManager**. User initialization classes are used during the initialization phase, while user action classes are used during the run. The run manager controls the flow of the program and manages the event loops within a run.

The visualization is enabled using **G4VisManager** if Geant4 is compile with OpenGL. Section 1.1 talks about the Geant4 installation options. The UI session or the interactive mode is enabled using **G4UImanager**. The UI session allows us to start runs, change certain parameters, open and manipulate a visualization session, and execute macros. Currently, the simulations need to be started via the command line. If you want to automatically start simulations on startup please uncomment and modify lines around line 220.

The verbosity class **EMMASteppingVerbose** is called which determines how much tracking informations for each event is printed to screen. The amount of screen outputs can be changed by using the terminal command **/track/verbose n**, where n is the level of verbosity.

Three input files are read in here as well:

1. **/UserDir/UserInput/beam.dat**
2. **/UserDir/UserInput/reaction.dat**
3. **/UserDir/UserInput/centralTrajectory.dat**

Examples of these files are given in section 5.

Details on Geant4 main() programs can be found in 2.1. How to Define the main() Program

7.2 User Initialization Classes

These are set to **G4RunManager** through **SetUserInitialization()** in the main program.

7.2.1 DetectorConstruction

This is the detector construction class. **EMMADetectorConstruction** depends on the following classes:

- **EMMADetectorConstructionMessenger**
- **EMMADriftChamber**
  - **EMMADriftChamberHit**
- **SpectrometerConstruction**

In the **EMMADetectorConstruction** class EMMA is constructed. Physical objects are called volumes in Geant4 and their properties include the geometry, material, position, any magnetic/electric fields and whether the volume is a sensitive detector or not. A sensitive detector is defined using **G4VSensitiveDetector** and is an abstract class which represents a detector.

Details on how to define properties of volumes can be found in 2.2. How to Define a Detector Geometry.

In **Construct()** of **EMMADetectorConstruction.cc** the target, degraders, the detector at the focal plane and Multiwire Proportional Chamber (MWPC) are defined. All materials are defined in **ConstructMaterials()** and the names of all volumes can be printed to terminal with **DumpGeometricalTree()**. Some of the properties, such as the target thickness or whether degraders and the MWPC are used, are defined in the **/UserDir/UserInput/targetDegraders.dat** and **/UserDir/UserInput/mwpc.dat** input files. The magnetic and electric rigidities of the magnetic and elec-

tric dipoles are calculated in the **CalculateScalingFactors()** method and use the **/UserDir/UserInput/centralTrajectory.dat** file input parameters.

The **EMMADetectorConstructionMessenger** implements commands to set values to variables needed to calculate these rigidities. When **EMMAapp** starts up these values are first read in from **/UserDir/UserInput/centralTrajectory.dat** in **EMMAapp.cc**, but can later be changed in the terminal. See section 3 for examples of how to change parameters using the command line. The **EMMADetectorConstructionMessenger** passes the input file and command line values to **EMMADetectorConstruction**.

The quadrupole magnets, electric dipoles, magnetic dipoles, slits and walls are constructed in **SpectrometerConstruction.cc**. The positions of the slits are read in from the **/UserDir/UserInput/slits.dat** file and are created in the **buildSlits()** method. The rest of the elements are created in the constructor **SpectrometerConstruction()**. The geometry and positioning of the elements seem complicated at first. This is why I created a macro called **visdebug.mac** so that I can display or hide any element that I want. This way you can figure out how each element is rotated and positioned. The electric and magnetic fields are implemented here by calling the **BDField** classes which in turn call the **mitray** fortran routines which calculate the fields at different positions located in the **fortran** folder.

If you make any changes to the names of the volumes in **EMMADetectorConstruction.cc** or **SpectrometerConstruction.cc** make sure the correct names are used in **EMMASteppingAction.cc** (described later). You can also check if there are any overlapping volumes by commenting out the **fCheckOverlaps=false;** line in the constructors located near the top of both classes. On start up of the program overlap information for each element will be printed to screen.

The detector at the focal plane is defined as a sensitive detector. The **EMMADriftChamber** class tells the simulation what to do when a particle enters the detector. It uses the information from the steps along the particle track. The kinetic energy, momentum, local position (w.r.t. the volume) and world position (w.r.t. the experimental hall) of each step in the detector are extracted in the **ProcessHits()** method. The **Print()** method in the **EMMADriftChamberHit** class then calls these values and outputs the detector hit information on to screen and a file called **/UserDir/Results/fp_beam.dat** or **/UserDir/Results/fp_reaction.dat**. Details on sensitive detectors can be found at 4.4.2. Sensitive detector.

7.2.2 EMMAPhysicsList

This is the physics list. **EMMAPhysicsList** depends on the following classes:

- **EMMAGeneralPhysics**
- **EMMAEMPhysics**
- **EMMAMuonPhysics**
- **EMMAHadronPhysics**
- **EMMAIonPhysics**
  - **EMMAIonPhysicsMessenger**
  - **EMMANuclearReactionProcess** NOT USED
    - **EMMANuclearReactionDataSet** NOT USED
  - **EMMANuclearReactionTwoBody** NOT USED
  - **G4ScreenedNuclearRecoil** NOT USED
    - **G4LindhardPartition** NOT USED
- **F04StepMax** NOT USED

This is where particles and their physics processes, i.e. interactions with matter, are defined. This is read in once when the program is started and nothing can be changed throughout the session. More information on particle types and physics processes can be found in 2.4. How to Specify Particles and 2.5. How to Specify Physics Processes.

The files that are not used are eventually going to simulate nuclear reactions including fusion evaporation reactions. Currently we fire the beam and recoil separately. The recoil kinematics are calculated using transfer reaction kinematics. The aim is to have Geant4 simulate the beam and reaction in the same process.

7.3 User Action Classes

These are set to **G4RunManager** through **SetUserAction()** in the main program.

7.4 EMMAPrimaryGeneratorAction

This is the primary generator action. **EMMAPrimaryGeneratorAction** depends on the following classes:

- **EMMAPrimaryGeneratorActionMessenger**

Here the particles are called/created and fired with a specified energy, direction and from a specified location. In the constructor **EMMAPrimaryGeneratorAction()** these variables are initialized. More details on primary generator actions can be found in 2.7. Geant4 General Particle Source. The **EMMAPrimaryGeneratorActionMessenger** class is called here. If the beam or reaction properties are changed in the command line the messenger will pass the values to the primary generator action. The constructor also reads in the values from the **/UserDir/UserInput/alphaSource.dat** file. If the first option in this file is set to 'YES' then an alpha particle is fired instead of the beam particle or recoil.

The methods **initializeBeamSimulation()**, **initializeBeamPreparation()** and **initializeReactionSimulation()** are called when the simulation run commands **/mydet/doBeam**, **/mydet/doPrepare** and **/mydet/doReaction** respectively are used. These methods open the output files and start the simulations using the method **G4RunManager::GetRunManager()->BeamOn(nEvents)**. The following output files are opened but not written to:

- **initializeBeamSimulation()**:
  - **/userDir/Results/fp_beam.dat**
  - **/userDir/Results/postTarget_beam.dat**
  - **/userDir/Results/postDegrader1_beam.dat**
- **initializeReactionSimulation()**:
  - **/userDir/Results/fp_reaction.dat**
  - **/userDir/Results/postTarget_reaction.dat**
  - **/userDir/Results/postDegrader1_reaction.dat**
- **initializeBeamPreparation()**:
  - **/UserDir/BeamSampling/beam.dat**

Descriptions of these files can be found in section 6. **initializeReactionSimulation()**, in addition to creating output files, also opens **/UserDir/BeamSampling/beam.dat** and reads in the energy, position and momentum of the beam particle at the reaction depth.

When the **BeamOn()** method of **G4RunManager** is invoked **GeneratePrimaries()** in the primary generator action is invoked first. Here the particle type is chosen and it is given a position, momentum and energy. These all depend on whether you're simulating beam particles, preparation beam particles for reactions, recoils or alpha particles.

For the preparation beam particles the variable **prepareBeam=true** and the following is done:

- a random reaction depth from a uniform distribution is selected
- the target thickness is updated to be equal to the reaction depth

If you're simulating the beam all the way through EMMA then the variable **simulateReaction=false**. In this case the target thickness is kept the original thickness. For both beam particles being prepared for a reaction and passing through EMMA the following is done:

- the lines in the **if (simulateReaction==false && !useAlphaSource)** condition are executed
- the beam particle is chosen
- the kinetic energy is chosen from a Gaussian distribution given by the mean and fractional energy spread (FWHM)
- the x and y emission location is chosen from a uniform distribution with the maximum distance given by the beam-spot diameter
- the z location is set to -0.1 m from the target
- the emission angle is chosen from a uniform distribution with the maximum angle calculated from the normalized transverse emittance (pi mm mrad)

The input parameters from **/UserDir/UserInput/beam.dat** are used here.

If you're simulating recoils the variable **simulateReaction=true** and the following is done:

- the lines in the **if (simulateReaction)** condition are executed

- the recoil particle is chosen
- for each recoil event a new kinetic energy, position and momentum direction of the beam at the reaction depth is called (these were read in from the **/UserDir/BeamSampling/beam.dat** file)
- a two body transfer reaction kinematics calculator **simulateTwoBodyReaction()** is called
- the reaction kinematics method calculates the energy and momentum direction of the recoil

The input parameters from **/UserDir/UserInput/reaction.dat** are used here.

If you're simulating alpha particles the variable **useAlphaSource=true** and the following is done:

- a 6Li(3+) particle is selected and the energy scaled by a factor 1.5 to simulate an ^{4}He particle (I think this is silly and should be changed)
- the energy from the **/UserDir/UserInput/alphaSource.dat** file is selected
- the emission location is at the origin which is at the target center
- the momentum direction is selected from a uniform angular distribution between 0 and a maximum angle

7.5 EMMASteppingAction

The stepping action class tracks the particles. This is done by invoking **G4Track** and **G4Step**. Explanations of these can be found in 5.1. Tracking.

In the **EMMASteppingAction()** constructor a ROOT histogram for dead hits is created. The ROOT file is created in **EMMAEventAction**. If a particle hits a wall or a slit the name of the volume hit is added to the histogram. This histogram is created and used only if ROOT is compiled.

The **UserSteppingAction()** method is called for each event. This method gets the position, kinetic energy and direction of each step along the particle track. For each step it finds/checks in which logical volume the particle is in.

For the reaction preparation beam the following is done:

- the lines in the **if (prepareBeam)** condition are executed
- the beam particle is tracked through the target
- the kinetic energy, position and direction of the last step point before the particle leaves the target is written to the **/UserDir/BeamSampling/beam.dat** file
- the event is terminated
- note: if the target is too thin, i.e. smaller than the step length, the beam particle might not see the target and pass through to the focal plane. The event will not write anything to file and won't be terminated. This results in fewer lines in the **/UserDir/BeamSampling/beam.dat** file and the program will crash if you run **/mydet/doReaction** with the same number of events. You can reduce the number of recoil events to equal the number of lines in the file to bypass this problem.

For the beam particles and recoils traveling through EMMA the variable **prepareBeam=false** and the following is done:

- the particles are tracked through the target
- the kinetic energy, angle and x and y position of the last step point before the particle leaves the target is written to the **/userDir/Results/postTarget_beam.dat** or **/userDir/Results/postTarget_reaction.dat** file
- the particles are tracked through the degrader if it is used
- the kinetic energy, angle and x and y position of the last step point before the particle leaves the degrader is written to the**/userDir/Results/postDegrader1_beam.dat** or **/userDir/Results/postDegrader1_reaction.dat** file
- if the particle hits a slit or wall then the name of the volume is added to the ROOT histogram and the event is terminated

7.6 EMMAEventAction

The event action class calls the **G4EventManager** and it gets the information at the beginning and end of each event. The **EMMAEventActionMessenger** is called in the **EMMAEventAction()** constructor. This passes the verbosity level, i.e. how much is printed

to screen for each event, to the program if it is changed in the command line. The constructor calls the **EMMADriftChamberHit** class to access the information of particles that entered the sensitive detector, in other words hit the focal plane. If ROOT is compiled then a ROOT file, some histograms and a tree are created. The ROOT file and tree are created in **EMMAAnalysisManager** and called into the constructor (this is for this simple purpose an unnecessarily complicated step).

The **BeginOfEventAction()** method is invoked at the start of every run. This just prints the event number.

The **EndOfEventAction()** method is invoked at the end of each run if not terminated earlier in the **UserSteppingAction()** method of **EMMASteppingAction.cc**. The information from the **EMMADriftChamberHit** class is called. The **Print()** method within **EMMADriftChamberHit** is invoked to print the energy and position of the particle at the focal plane to screen and writes the energy, angle and focal plane position to the **/UserDir/Results/fp_beam.dat** or **/UserDir/Results/fp_reaction.dat** file. The values are also added to the ROOT histograms and tress.