Running the CHANCE LeadBlock Example

The chance_g4sim package is setup to automatically produce the required detector geometry for you. That way the user only has to create a JSON table when they want to add blocks of different materials.

In this example a lead block is added between the top and bottom detector planes and events are generated with an exposure time of 1 hour.

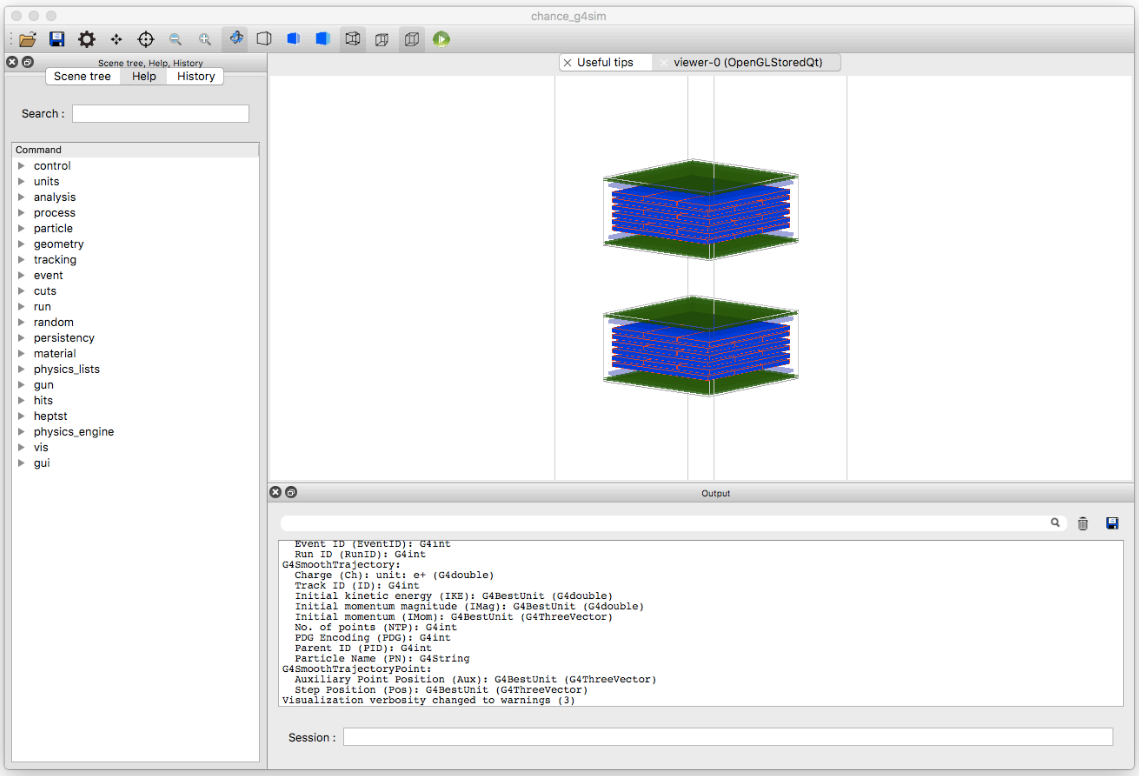
1. Running the simulation package

Once the code has been built, you can run the chance simulation package from any directory by sourcing the setup in your build directory.

```
$ source build/setup.sh
$ chance_g4sim -h
```

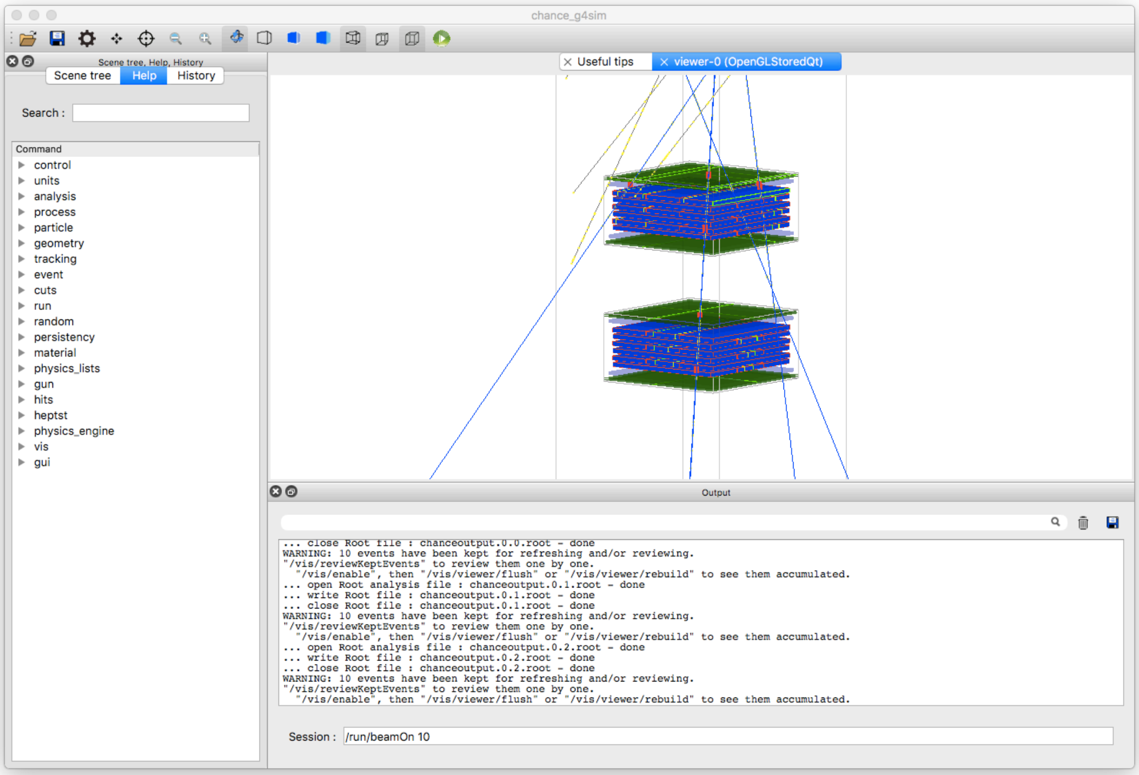
To run interactively and view the loaded geometry, the '-*i*' flag can be used at runtime to bring up an OpenGL viewer.

```
$ chance_g4sim -i
```



```
  Event ID (EventID): G4int
  Run ID (RunID): G4int
G4SmoothTrajectory:
  Charge (Ch): unit: e+ (G4double)
  Track ID (ID): G4int
  Initial kinetic energy (IKE): G4BestUnit (G4double)
  Initial momentum magnitude (IMag): G4BestUnit (G4double)
  Initial momentum (IMom): G4BestUnit (G4ThreeVector)
  No. of points (NTP): G4int
  PDG Encoding (PDG): G4int
  Parent ID (PID): G4int
  Particle Name (PN): G4String
G4SmoothTrajectoryPoint:
  Auxiliary Point Position (Aux): G4BestUnit (G4ThreeVector)
  Step Position (Pos): G4BestUnit (G4ThreeVector)
Visualization verbosity changed to warnings (3)
```

Some sample events can be created by typing the command "/run/beamOn 100" in the G4 interactive terminal box.



```
... close Root file : chanceoutput.0.0.root - done
WARNING: 10 events have been kept for refreshing and/or reviewing.
"/vis/reviewKeptEvents" to review them one by one.
  "/vis/enable", then "/vis/viewer/flush" or "/vis/viewer/rebuild" to see them accumulated.
... open Root analysis file : chanceoutput.0.1.root - done
... write Root file : chanceoutput.0.1.root - done
... close Root file : chanceoutput.0.1.root - done
WARNING: 10 events have been kept for refreshing and/or reviewing.
"/vis/reviewKeptEvents" to review them one by one.
  "/vis/enable", then "/vis/viewer/flush" or "/vis/viewer/rebuild" to see them accumulated.
... open Root analysis file : chanceoutput.0.2.root - done
... write Root file : chanceoutput.0.2.root - done
... close Root file : chanceoutput.0.2.root - done
WARNING: 10 events have been kept for refreshing and/or reviewing.
"/vis/reviewKeptEvents" to review them one by one.
  "/vis/enable", then "/vis/viewer/flush" or "/vis/viewer/rebuild" to see them accumulated.
```

Session : /run/beamOn 10

Press CTRL+C in your original terminal to exit this program.

2. Creating a JSON Table

JSON tables are used to define new geometry or runtime options without having to recompile. Some general documentation on the JSON tables can be found here: http://cosmicraysim.readthedocs.io/en/latest/

To load a new table in at runtime, first we have to write a ".geo" file with all our information in. The following table will create a block of lead of size 10 x 10 x 10 cm and place it at the origin in the world volume (between our two detector planes)

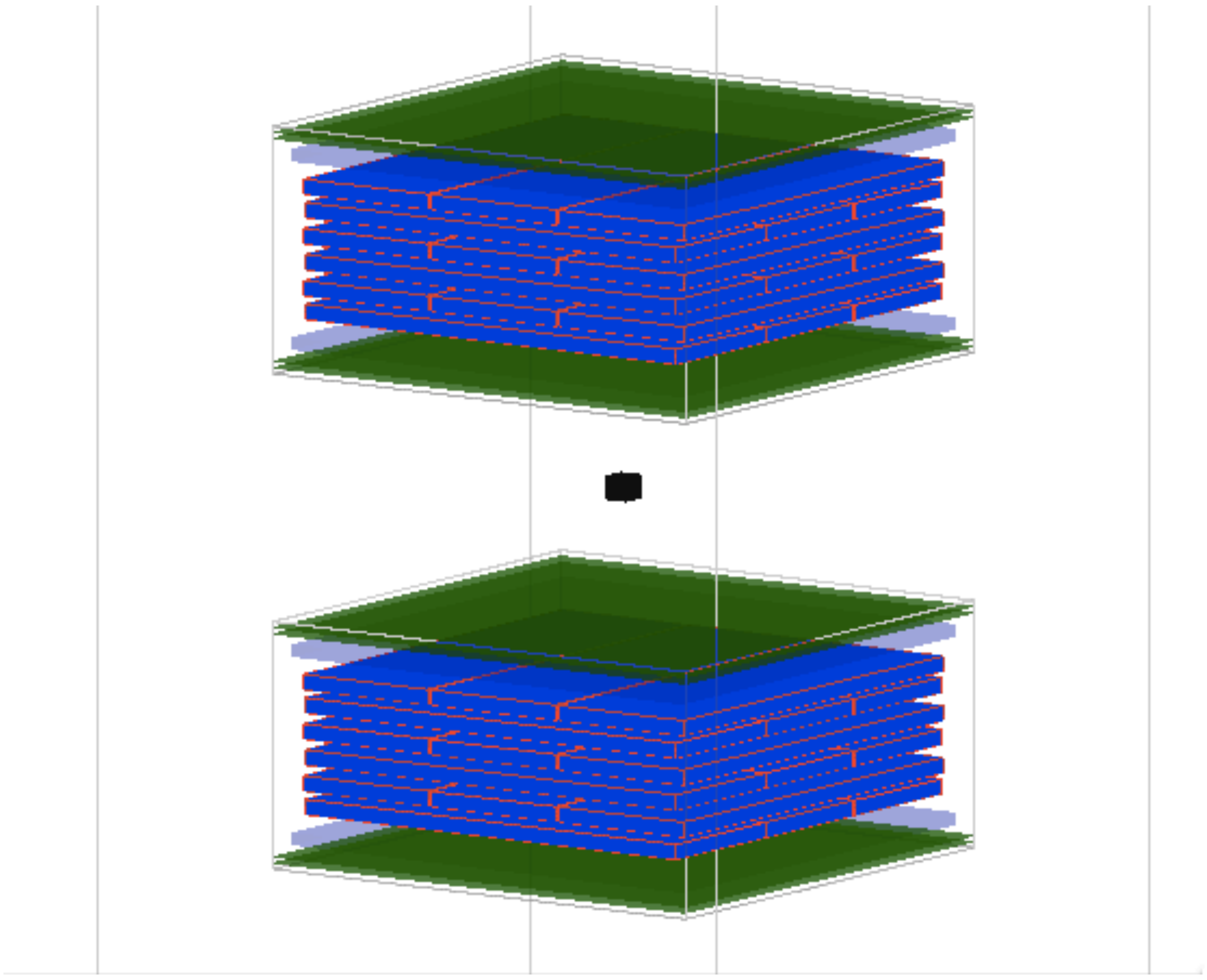
leadblock_table.geo

```
{
  name: "GEO",
  index: "leadblock"
  type: "box",
  size: ["10*cm","10*cm","10*cm"]
  position: ["0.0*m","0.0*m","0.0*m"]
  rotation: [0.0, 0.0, 0.0]
  material: "G4_Pb"
  mother: "world"
}
```

This table is also provided for you inside cosmicraysim-*chance/examples/chance/leadblock/leadblock_table.geo*

To load this table in at run time, we just have to specifiy the table with the "-g" flag.

```
$ chance_g4sim -g leadblock_table.geo -i
```



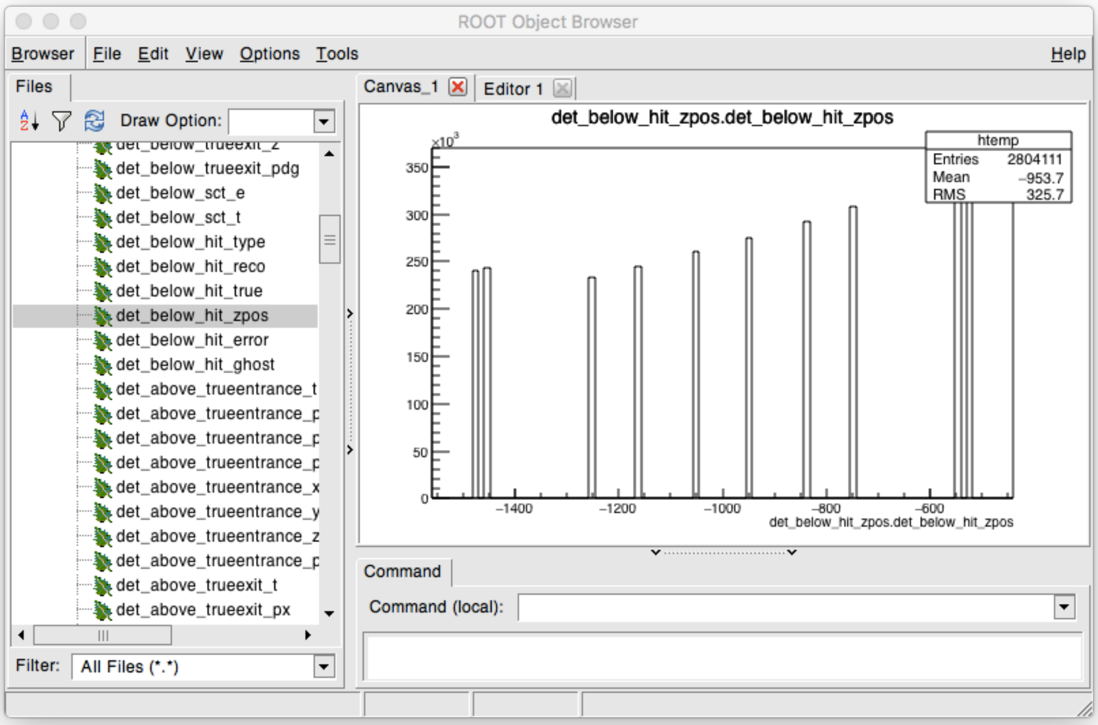
If you wanted to load multiple tables at once, you could add them all to a single file, or you could put each one in a different file and use the “-g” flag multiple times to lead in each file.

3. Generating an hour of exposure

The chance_g4sim package is already setup to generate cosmic ray from a planes at the top of the world volume with a 3 x 3 m plane. The “-t” flag can be used to specify how many seconds you want to generate.

```
$ cosmicraysim -g leadblock_table.geo -t 3600
```

This will produce an output ROOT file called *chanceoutput.0.0.root.* This contains hit positions for both RPC and Drift chambers in std::vectors. The hit type says what type of detector was hit. E.g. RPC X.



If you want to rename the file, or give it a RUNID if you are running in parallel, you can use the "-o" and "-r" flags. The following command will generate an output file called "myleadblock.10.0.root"

```
$ cosmicraysim -g leadblock_table.geo -t 3600 -o myleadblock --run 10
```

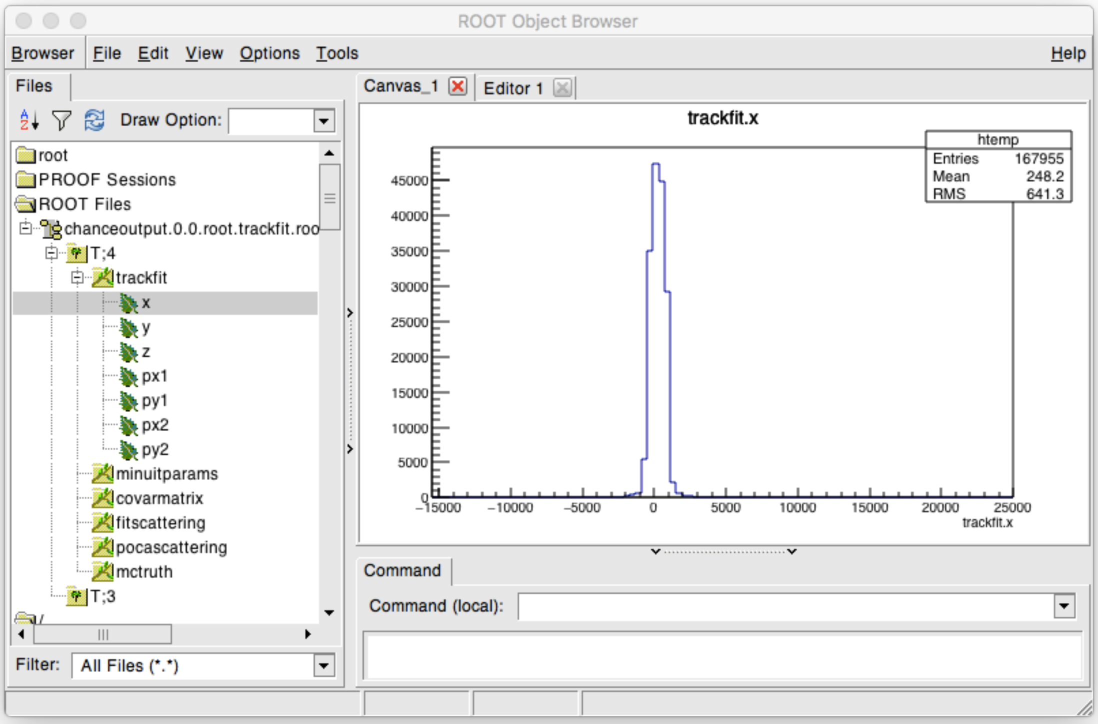
4. Running the track fitter

The track fit code then needs to be ran on these outputs to generate muon scatter vertices. This should already be built and can be run using the command

```
$ chance_trackfit -i chanceoutput.0.0.root
```

This will take quite a while, I'm working on speeding up the trackfitter…..
The output will be a file called chanceoutput.0.0.root.trackfit.root

The efficiency of the trackfitter is about 50%, largely due to some tracks not leaving 10 hits (4 RPC + 6 Drift) in the above or below detector.



5. Running the discriminator

At this point you can either use the Bristol MuonAnalysisFW code to process the trackfit output (without momentum!) or you can play around with our own version. Our version can be ran using

```
$ chance_discr -i chanceoutput.0.0.root.trackfit.root
```

The output file will be called "trackfit.discriminator.root", and will contain several TH3D histograms with the metric discriminator values saved into each bin. The draw.py example script in the examples folder will draw slices of this grid based on the angular statistics reconstruction algorithm.