

MINI-HALO USER'S MANUAL

Supervisor: Dr. Clarence Virtue
Author: Kaelan Renault

In Fulfillment of the Requirements for
Arthur B. McDonald Canadian Astroparticle Physics
Research Institute - Cross Disciplinary Internship

August 28th, 2020

Table of Contents

| | |
|---|-----------|
| Table of Contents | 1 |
| 1 Introduction | 2 |
| 2 Running a Simulation | 2 |
| 2.1 Connecting to Nearline | 2 |
| 2.2 Setting up simulation parameters | 3 |
| 2.3 Compiling the simulation | 6 |
| 2.4 Executing the simulation | 7 |
| 2.5 Visualizing the simulation | 9 |
| 3 Parameters | 12 |
| 3.1 GraphiteThickness / HDPETthickness | 13 |
| 3.2 GraphiteMaterials / HDPEMaterials / LeadMaterials | 14 |
| 3.3 IncludeConcreteShield / IncludeCalibrationTubes | 19 |
| 3.4 CheckOverlaps | 21 |
| 4 Changing the Code | 22 |
| 4.1 Project layout | 22 |
| 4.2 Project standards | 24 |
| 5 ROOT Analysis | 25 |
| 5.1 Accessing implemented ROOT graphs | 25 |

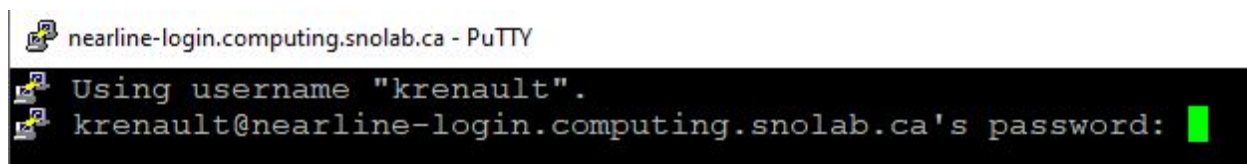
1 Introduction

Unless significant changes to the code and environment are made after August 2020, this user's manual should be all the information required to use the Mini-HALO Monte Carlo simulation. If this document is not satisfactory, other helpful reference documents might include: *GEANT4 Implementation of Mini HALO* by Chandler Ross (2019), *He-3 Proportional Counter Simulation Report* by Gareth Smith (2017), and the MINI HALO PRELIMINARY DESIGN technical document for the detector (2018). Dr. Clarence Virtue should be the first point of contact for questions on the project as a whole, and Remington Hill should be the first point of contact for specific technical questions regarding the simulation or ROOT analysis.

2 Running a Simulation

2.1 Connecting to Nearline

The first thing you might want to do when you're starting to work on this project is to run the simulation. To do so you'll need to connect to the Nearline computing cluster, which is not only where you can find a copy of the project, but also where the required versions of Geant4, ROOT, C++, and Java are already installed. Before doing anything on Nearline, read the documentation on the SNOLAB internal site for protocols and proper usage of the system. To connect, you'll need to turn on your VPN connection to SL-GEN using your SNOLAB credentials. Once that's done, SSH to the cluster using whatever SSH client works for you. While I was working on the project I was using PuTTY to connect and Vcxsrv X11 to render the visualization.



```
nearline-login.computing.snolab.ca - PuTTY
Using username "krenault".
krenault@nearline-login.computing.snolab.ca's password: 
```

Protip: To make sure Vcxsrv is running, try the command 'xclock' on Nearline

2.2 Setting up simulation parameters

Now that you've connected to the cluster, you should be looking at a typical linux command line prompt.

```
Last login: Tue Aug 18 13:36:12 2020 from vpnout.snoLab.ca
=====
Welcome to the SnoLab Internal Compute cluster!
=====

lmod is automatically replacing "intel/2016.4" with "gcc/7.3.0".

Due to MODULEPATH changes, the following have been reloaded:
  1) openmpi/2.1.1

The following have been reloaded with a version change:
  1) gcccore/.5.4.0 => gcccore/.7.3.0

[krenault@nearline-login ~]$
```

To move to the directory containing the Mini-HALO code, run the following command:

```
cd /project/halo/mini_halo
```

That directory should contain the `miniHALO`, and `miniHALO-build` directories, along with backups or anything else pertaining to the project.

```
[krenault@nearline-login mini_halo]$ ls
backups  miniHALO  miniHALO-build
```

To run the simulation, you want to be in the `miniHALO-build` directory.

```
cd miniHALO-build/
```

This directory contains all of the different macro files that can run the simulation, along with the Makefiles, visualization files, analysis files, and everything else that is relevant to the code at or after compiling and executing the simulation. Source code is in `miniHALO` (In the `miniHALO-build` parent directory).

```
[krenault@nearline-login miniHALO-build]$ ls
CMakeCache.txt          miniHALO                      NeutronsInLead.mac
CMakeFiles              miniHALO_break_through_neutrons.mac  test-analysis-script.cc
cmake_install.cmake     miniHALO.in                   test-output.root
G4Data0.heprep          miniHALO.out                   test.root
G4History.macros        miniHALO_shielded_break_through_neutrons.mac  user.properties
HepRApp.jar             miniMAKE                       vis_kaelan.mac
Makefile                miniRUN                         vis.mac
```

I'll use my own `vis_kaelan.mac` file as an example for running the simulation here, but feel free to copy it and make your own macro to simulate some other aspects of the detector. I'm also going to use Vim, but you can use emacs or nano or any other text editor you prefer to work on these files.

```
vi vis_kaelan.mac
```

Once I have the file open, it looks like this:

```
***: I'm writing my own macro based on the vis_tom.mac file that Chandler was using
*   in summer 2019. Since I don't really know what I'm doing with these files some
*   of the commands might be useless in the current version of Mini-HALO.

/control/verbose 0
/control/saveHistory

*/tracking/verbose 1 /*<-this one for full output
/tracking/verbose 0 /*<- this one for no output
/run/verbose 0
/run/initialize #*****

/gps/particle neutron
*/gps/ene/mono 10 MeV
/gps/pos/type Volume
/gps/pos/shape Sphere
/gps/pos/centre 0 0 0 mm
/gps/pos/radius 115 cm
/gps/pos/confine p_lead_box
/gps/ang/type iso

/gps/ene/type Arb
/gps/hist/type arb

* The Cf-252 neutron energy spectrum was taken from
* Figure 2 of A.B. Smith and P.R. Fields,
* Phys Rev 108 (1957) 411.
*
*
*          Energy (MeV)  Rel. Yield
/gps/hist/point 0.03791  32.71076
/gps/hist/point 0.08301  45.89684
/gps/hist/point 0.19830  59.07265
/gps/hist/point 0.34993  68.10622
/gps/hist/point 0.60544  74.34164
/gps/hist/point 0.81070  76.17980
/gps/hist/point 1.08555  73.89820
/gps/hist/point 1.55102  64.25226
/gps/hist/point 1.89664  55.40119
/gps/hist/point 2.39728  45.95365
/gps/hist/point 2.82949  37.80812
/gps/hist/point 3.50389  27.66813
/gps/hist/point 4.40363  17.72709
/gps/hist/point 5.04448  12.47039
/gps/hist/point 5.61632  8.97969
/gps/hist/point 6.24062  6.12068
/gps/hist/point 7.02125  3.73887
/gps/hist/point 8.84360  1.12916

* Linear interpolation between the data points is required to obtain
* the smooth energy spectrum
/gps/hist/inter Lin

/miniHALO/DetectorConstruction/CheckOverlaps 0
```

```
/miniHALO/DetectorConstruction/CheckOverlaps 0  
  
/miniHALO/DetectorConstruction/update  
  
# HepRApp commands  
#/vis/scene/create  
#/vis/open HepRepFile  
#/vis/viewer/flush  
#/vis/drawVolume  
#/vis/scene/add/trajectories  
#/vis/scene/endOfEventAction accumulate  
  
/tracking/storeTrajectory 1  
  
/run/beamOn 50000
```

The dark blue is just comments, so don't be too concerned if you can't make them out in the screenshots. The `/tracking/` and `/run/` commands I pulled from Chandler's old macro on HaloShift and I don't know for sure that they need to be there, but they haven't caused me any issues and might be critically important so I would include them in your own macro. The `/gps/` commands are from Geant4's General Particle Source, and you can find documentation detailing all the options online. The commands in my macro generate neutrons confined to the lead block in the detector, with initial energies pulled from the energy spectrum defined in the macro (which is the energy spectrum of Cf-252). Other macros within this directory generate particles differently (for instance on a plane to one side of the detector for breakthrough neutrons) and you can use those, in addition to this macro, to help guide you on how to specify the way you want to generate particles in the detector.

The `/miniHALO/` commands are the parameters that we've defined within the code, described in detail in section 3 of this manual.

Protip: Always keep `/miniHALO/DetectorConstruction/update` AFTER the `/miniHALO/` parameters we talk about in section 3 in your macro.

Lastly, `/run/beamon` specifies the number of particles to simulate. I would start with 10 if you're just confirming you can get everything working for the simulation. More than 10 is usually only necessary for collecting results, otherwise it only slows down simulations when updating the geometry or testing other changes to the code.

2.3 Compiling the simulation

Protip: You only need to compile the simulation if you modify one of the files in /project/halo/mini_halo/miniHALO/ otherwise you can skip to section 2.4

Once you've made some changes to the source code that you want to run (always run to make sure everything works after you've made changes) you'll need to compile the code. I'll run through the manual steps needed to compile, but I've written a script that can be submitted to slurm that does everything automatically (miniMAKE in miniHALO-build) by running the following command from the miniHALO-build directory.

```
sbatch miniMAKE
```

Once the slurm job returns (check `squeue --user=krenault`, replace my username with yours.) check the output which will be contained in a slurm-[jobID] text file to make sure it looks something like this (screenshot of the end of the file). You are safe to assume that there weren't any issues if it says "[100%] Linking CXX executable miniHALO [100%] Built target miniHALO" at the end of the file.

```
In file included from /cvmfs/soft.computecanada.ca/easybuild/software/2017/avx512/Compiler/gcc7.3/root/6.14.04/include/TClass.h:26:0,
      from /cvmfs/soft.computecanada.ca/easybuild/software/2017/avx512/Compiler/gcc7.3/root/6.14.04/include/TKey.h:18,
      from /cvmfs/soft.computecanada.ca/easybuild/software/2017/avx512/Compiler/gcc7.3/root/6.14.04/include/TBasket.h:28,
      from /cvmfs/soft.computecanada.ca/easybuild/software/2017/avx512/Compiler/gcc7.3/root/6.14.04/include/ROOT/TIOFeatures.hxx:14,
      from /cvmfs/soft.computecanada.ca/easybuild/software/2017/avx512/Compiler/gcc7.3/root/6.14.04/include/TTree.h:29,
      from /project/halo/mini_halo/miniHALO/include/miniHALOAnalysis.hh:25,
      from /project/halo/mini_halo/miniHALO/src/miniHALOSTeppingAction.cc:5:
/cvmfs/soft.computecanada.ca/easybuild/software/2017/avx512/Compiler/gcc7.3/root/6.14.04/include/TObjString.h: In member function 'void TObjString::SetString(const char*)':
/cvmfs/soft.computecanada.ca/easybuild/software/2017/avx512/Compiler/gcc7.3/root/6.14.04/include/TObjString.h:46:41: warning: declaration of 's' shadows a global declaration [-Wshadow]
      void      SetString(const char *s) { fString = s; }
      ^
In file included from /cvmfs/soft.computecanada.ca/easybuild/software/2017/avx512/Compiler/gcc7.3/clhep/2.4.1.0/include/CLHEP/Units/PhysicalConstants.h:43:0,
      from /cvmfs/soft.computecanada.ca/easybuild/software/2017/avx512/Compiler/gcc7.3/clhep/2.4.1.0/include/CLHEP/Vector/RotationX.icc:11,
      from /cvmfs/soft.computecanada.ca/easybuild/software/2017/avx512/Compiler/gcc7.3/clhep/2.4.1.0/include/CLHEP/Vector/RotationX.h:283,
      from /cvmfs/soft.computecanada.ca/easybuild/software/2017/avx512/Compiler/gcc7.3/clhep/2.4.1.0/include/CLHEP/Vector/Rotation.h:30,
      from /project/halo/remington/Programs/geant4.10.06.p02-install/include/Geant4/G4RotationMatrix.hh:40,
      from /project/halo/remington/Programs/geant4.10.06.p02-install/include/Geant4/G4VPhysicalVolume.hh:48,
      from /project/halo/remington/Programs/geant4.10.06.p02-install/include/Geant4/G4Step.hh:60,
      from /project/halo/mini_halo/miniHALO/include/miniHALOAnalysis.hh:10,
      from /project/halo/mini_halo/miniHALO/src/miniHALOSTeppingAction.cc:5:
/cvmfs/soft.computecanada.ca/easybuild/software/2017/avx512/Compiler/gcc7.3/clhep/2.4.1.0/include/CLHEP/Units/SystemOfUnits.h:151:28: note: shadowed declaration is here
      static constexpr double s = second;
      ^
[100%] Linking CXX executable miniHALO
[100%] Built target miniHALO
Wed Aug 19 11:17:05 EDT 2020
```


Skip to section 2.4 if the script I wrote is still working, the following commands are the manual steps to replace the script.

Move to the miniHALO directory like so:

```
cd /project/halo/mini_halo/miniHALO/
```

Then you can CMake the directory:

```
cmake .
```

Change directory back to miniHALO-build:

```
cd ../miniHALO-build
```

Clean the old makefile, just for safety:

```
make clean
```

Then make the directory:

```
make
```

Again, you can confirm the project compiled if it looks like this after you `make`:

```
[100%] Linking CXX executable miniHALO
[100%] Built target miniHALO
```

2.4 Executing the simulation

Now you are finally ready to simulate the detector! Again, there is a macro that I wrote to submit to slurm, but it's currently set up to run `vis_kaelan.mac`, so change it to your file name. Run the following from the miniHALO-build directory.

```
sbatch miniRUN
```

Once the slurm job returns (check `squeue --user=krenault`, replace my username with yours.) check the output which will be contained in a `slurm-[jobID]` text file to make sure it looks something like this (screenshot of the end of the file):

Running the simulation without the script is comparatively easier this time. Run the following command from the miniHALO-build directory, replacing my macro name with yours:

```
./miniHALO vis_kaelan.mac
```

You should be able to determine if there was an issue the exact same way as above with the output of the slurm job, but the output will all be in the terminal with this method.

2.5 Visualizing the simulation

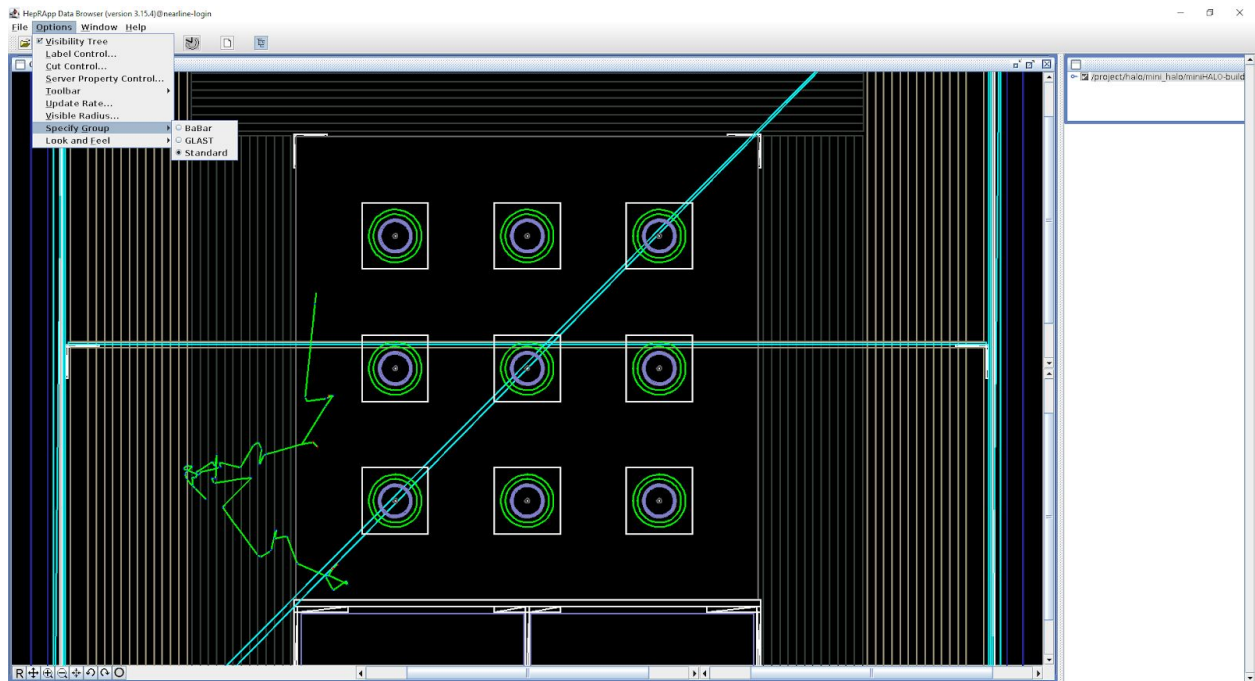
To visualize the simulation, you first need to run it with these commands specified in the macro file:

```
# HepRApp commands
/vis/scene/create
/vis/open HepRepFile
/vis/viewer/flush
/vis/drawVolume
/vis/scene/add/trajectories
/vis/scene/endOfEventAction accumulate
```

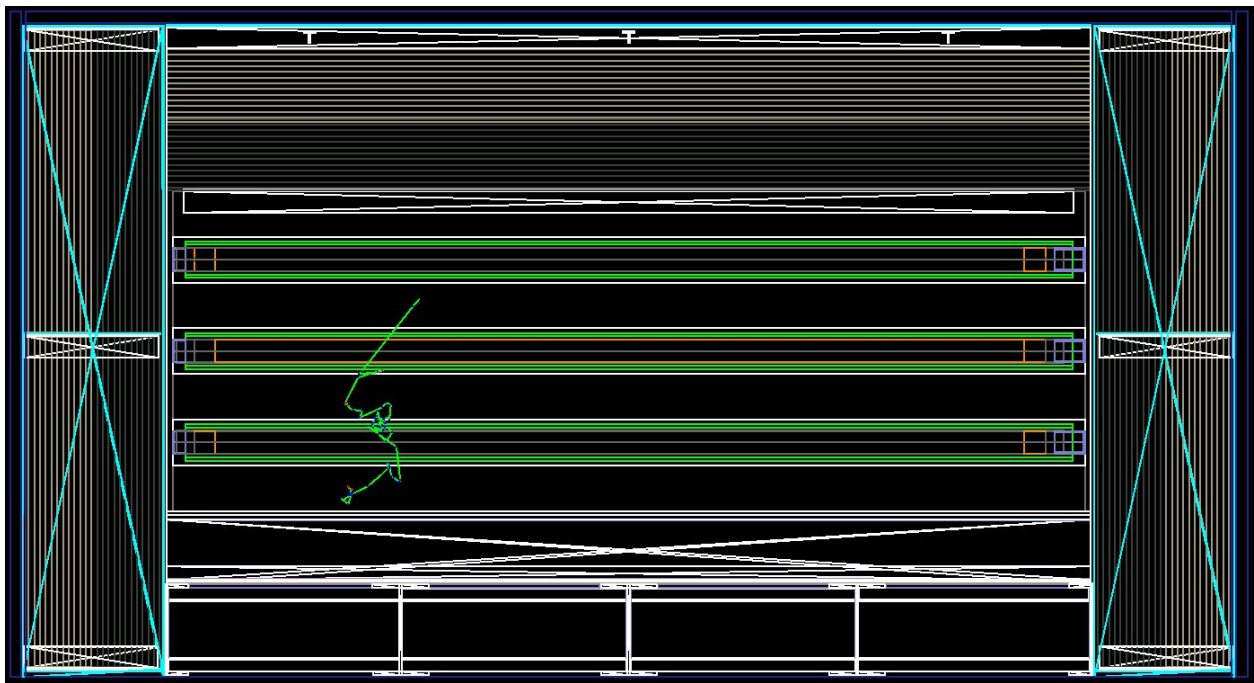
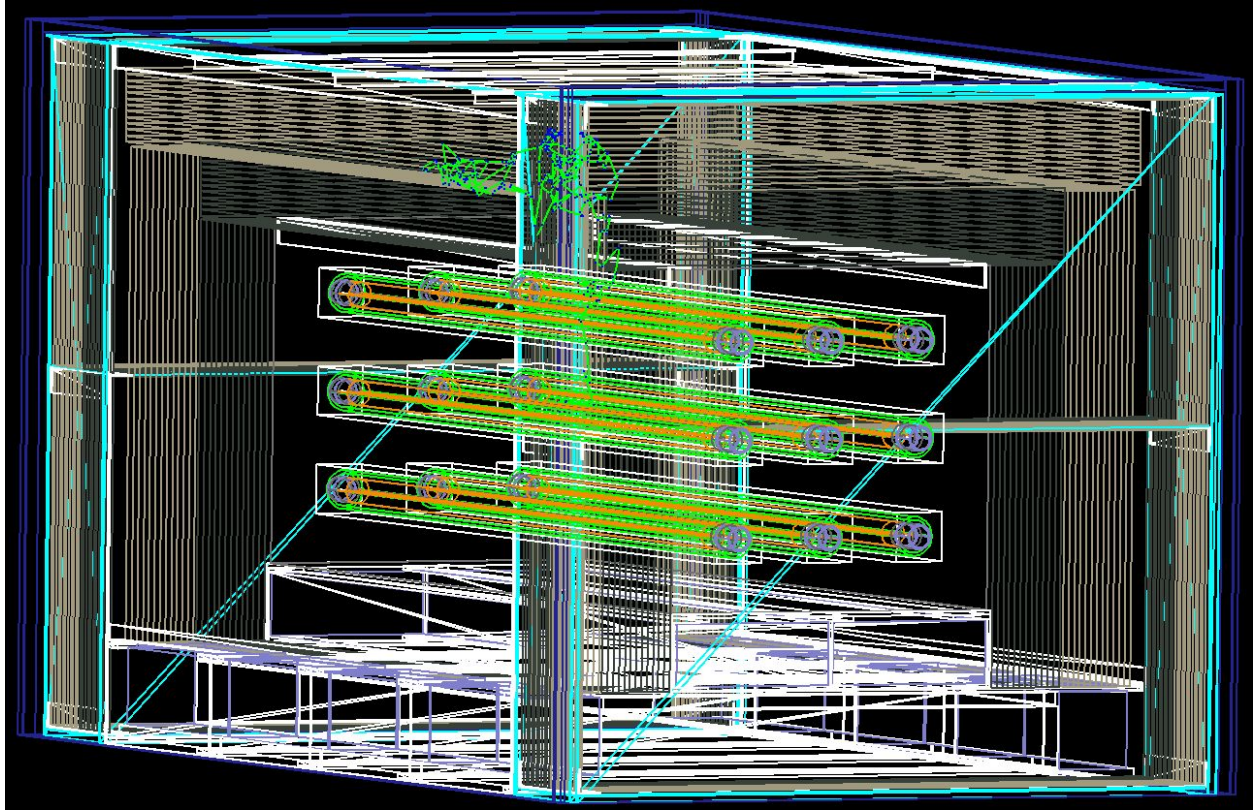
Once you've run a simulation with the above parameters enabled (See section 2.4), there will be a `G4Data0.heprep` file in the miniHALO-build directory. To view the visualization you'll need to make sure your X11 server (or equivalent) is enabled and run the following command:

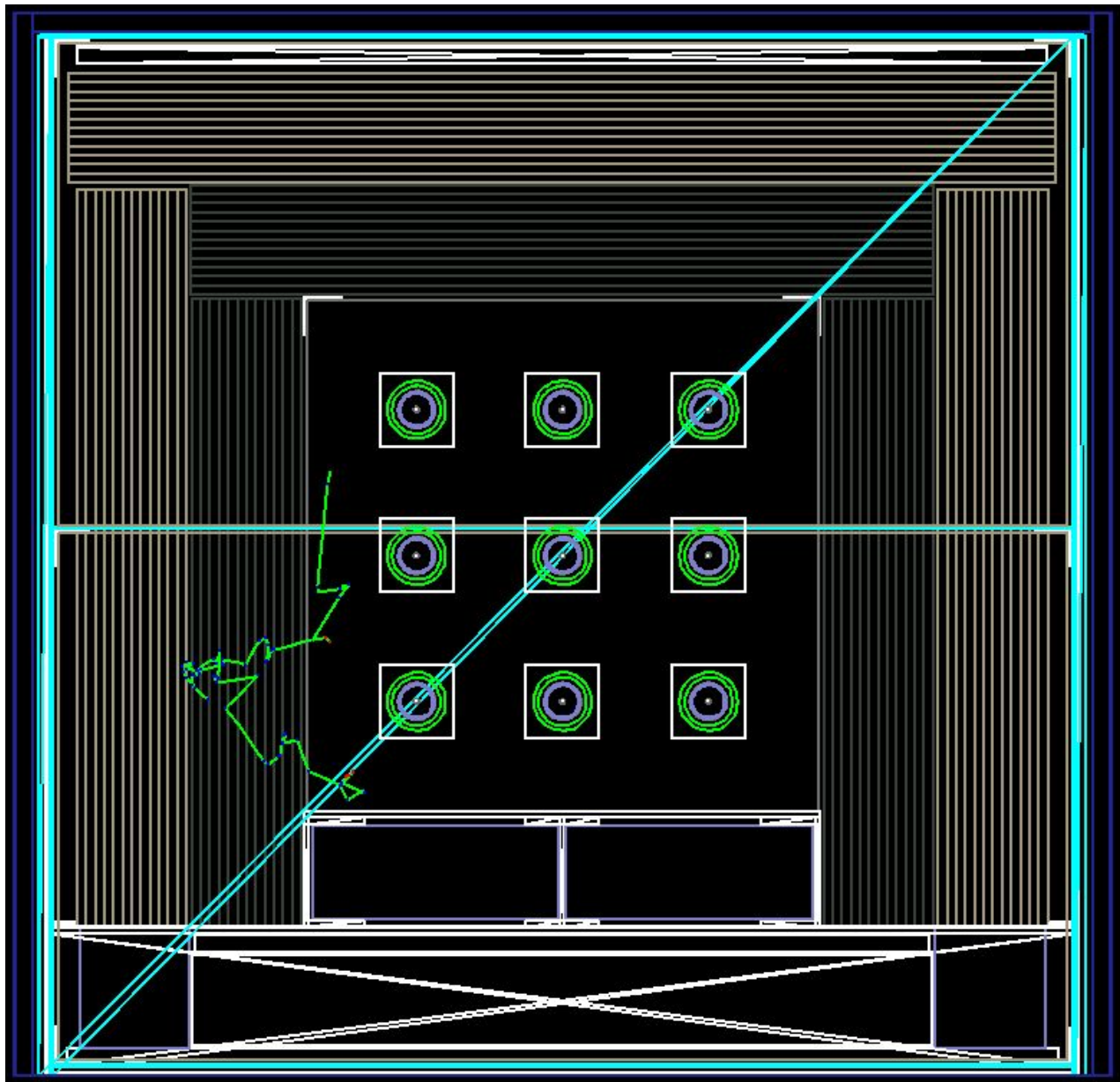
```
java -jar HepRApp.jar -file G4Data0.heprep
```

The visualization should load looking something like the following image when you first open it:



I'll leave learning to use HepRApp out of the scope of this document, but you can use some of the buttons in the bottom left corner to adjust your view and the sliders along the X and Y axis to rotate or move along that axis. The visualization should look something like the following images where you can see it from different angles:





3 Parameters

There are a number of parameters that can be set in the macro file used to set-up the simulation that we've defined. All the parameters have a default value, so none of them need to be explicitly called for everything to run smoothly. They also all have some error handling, so look out for error messages in the output if you tried to make a change and it's not taking. That probably means your input isn't valid and the simulation used the default values for that parameter. All these parameters start with `/miniHALO/DetectorConstruction/` in the macro file.

3.1 GraphiteThickness / HDPETthickness

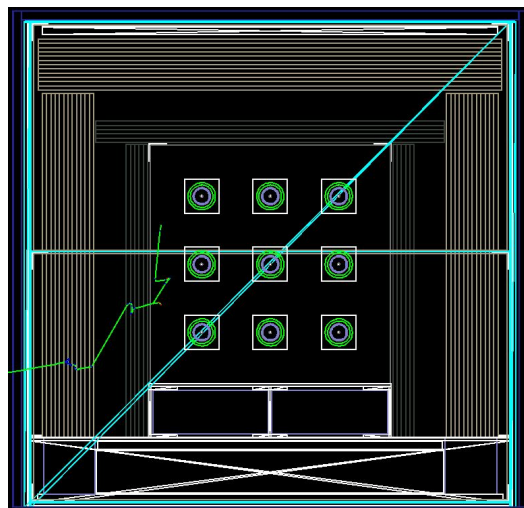
The GraphiteThickness and HDPETthickness parameters were developed before the material parameters of section 3.2, and should only be used if the materials parameters are not being specified. The reason for this is that the thickness parameters prevent some volumes from being instantiated (those furthest away from the center of the detector), and the materials parameters don't register that those volumes won't actually get created. This means that if you specify graphite thickness of 4.5 inches and specify the graphite material to be Aluminum-graphite-HDPE; expecting three 1.5 inch layers of the different materials, you will be disappointed. You will end up with 2 inches of aluminum, 2 inches of graphite, and 0.5 inches of HDPE.

Protip: Use the Materials commands instead of the Thickness commands unless you really know what you're doing and you're sure this is what you want.

That being said, the parameters are easy to use, you just specify a floating point number in 0.5 inch increments from 0 to 6. When the simulation is run, it will create the layers from the inside of the detector outwards, stopping at the desired thickness.

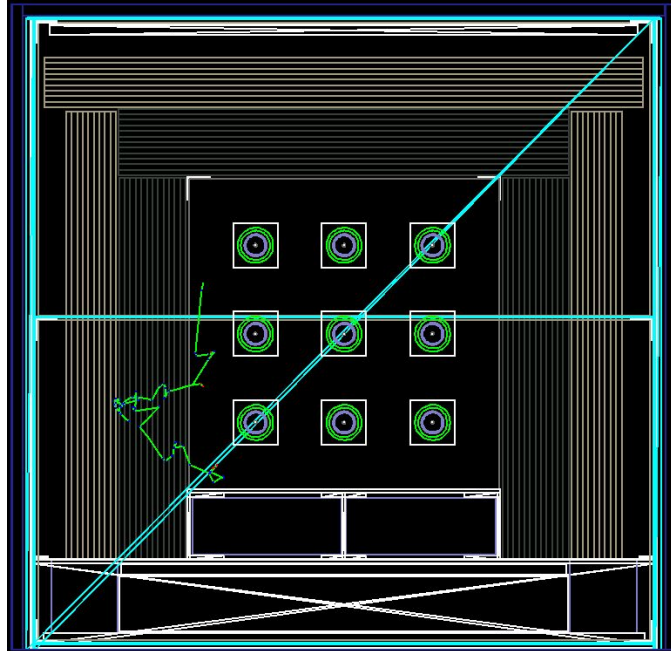
```
/miniHALO/DetectorConstruction/GraphiteThickness 2.5
```

Given the above parameters, the visualization should look like this (When compared to the last image in section 2.5, which is the same configuration without any parameters specified):



Similarly, running the simulation with the following parameter specified results in the following configuration for the detector.

```
/miniHALO/DetectorConstruction/HDPEThickness 4.5
```



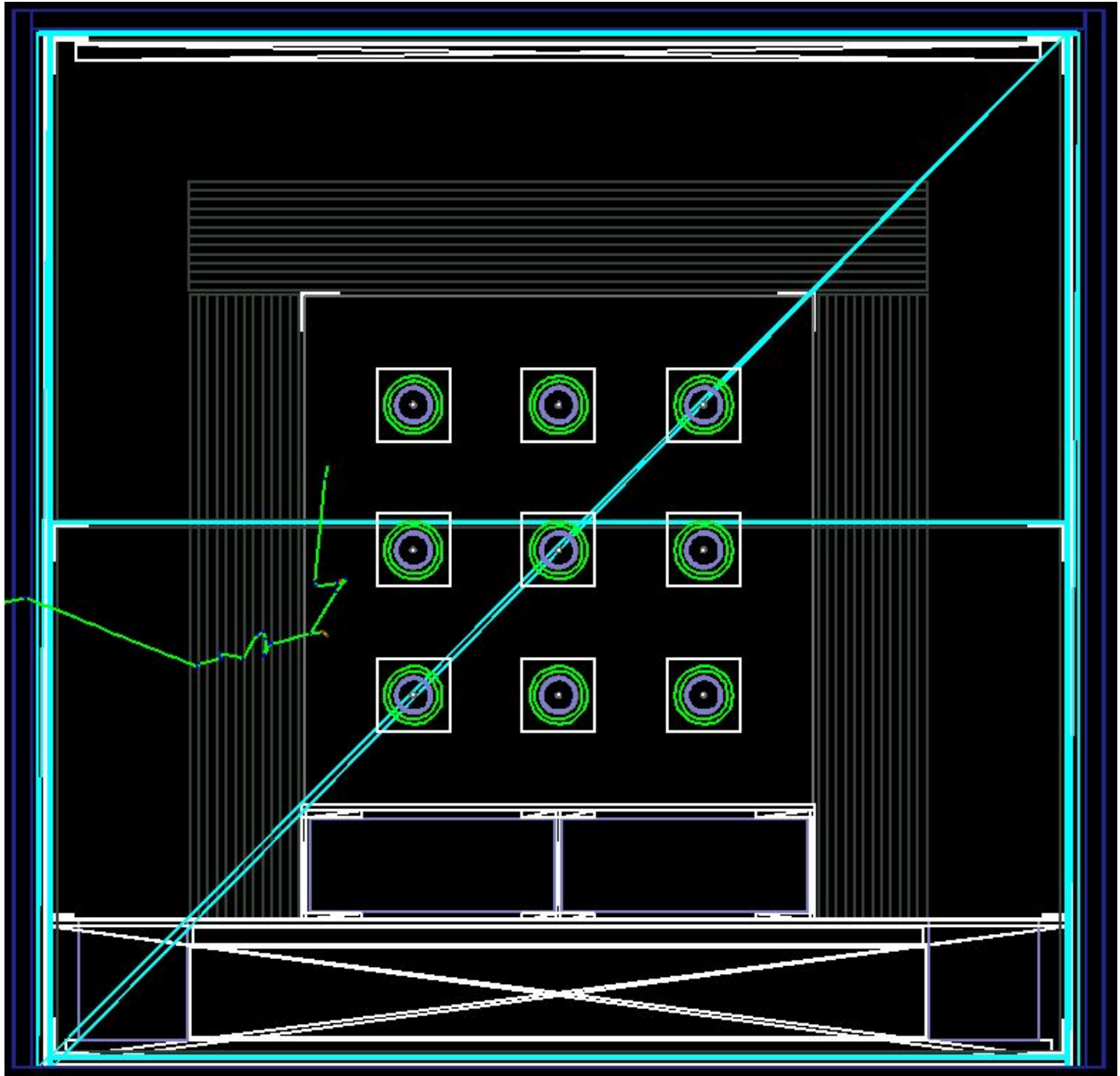
3.2 GraphiteMaterials / HDPEMaterials / LeadMaterials

The three materials parameters are much more interesting than the parameters in section 3.1, and everything that can be done using the thickness parameters can also be done with these parameters.

Both the graphite and HDPE material parameters work in exactly the same way. You optionally provide a side (or many sides) of the detector and provide a material (or many materials). This will apply that configuration of materials to the sides specified. If no sides are specified, it will apply to all 5 sides of the detector. These parameters can be specified multiple times so that different material configurations can be applied to different sides of the detector. It's worth mentioning that the sides don't need to be specified in any particular order. It's also worth mentioning that the number of materials needs to be a factor of 12, so that they can be split evenly across the 12 layers of the material. That means you can specify 0 or more sides per parameter call and 1, 2, 3, 4, 6, or 12 materials per parameter. Sides and materials are all separated by hyphens (And not spaces). I'll walk through a number of examples to showcase the different capabilities of these parameters.

Example #1: All HDPE to air. If we don't specify any sides and only specify one material, all sides of the detector will have all their layers converted into the new material (in this case Air, which renders as invisible in the detector).

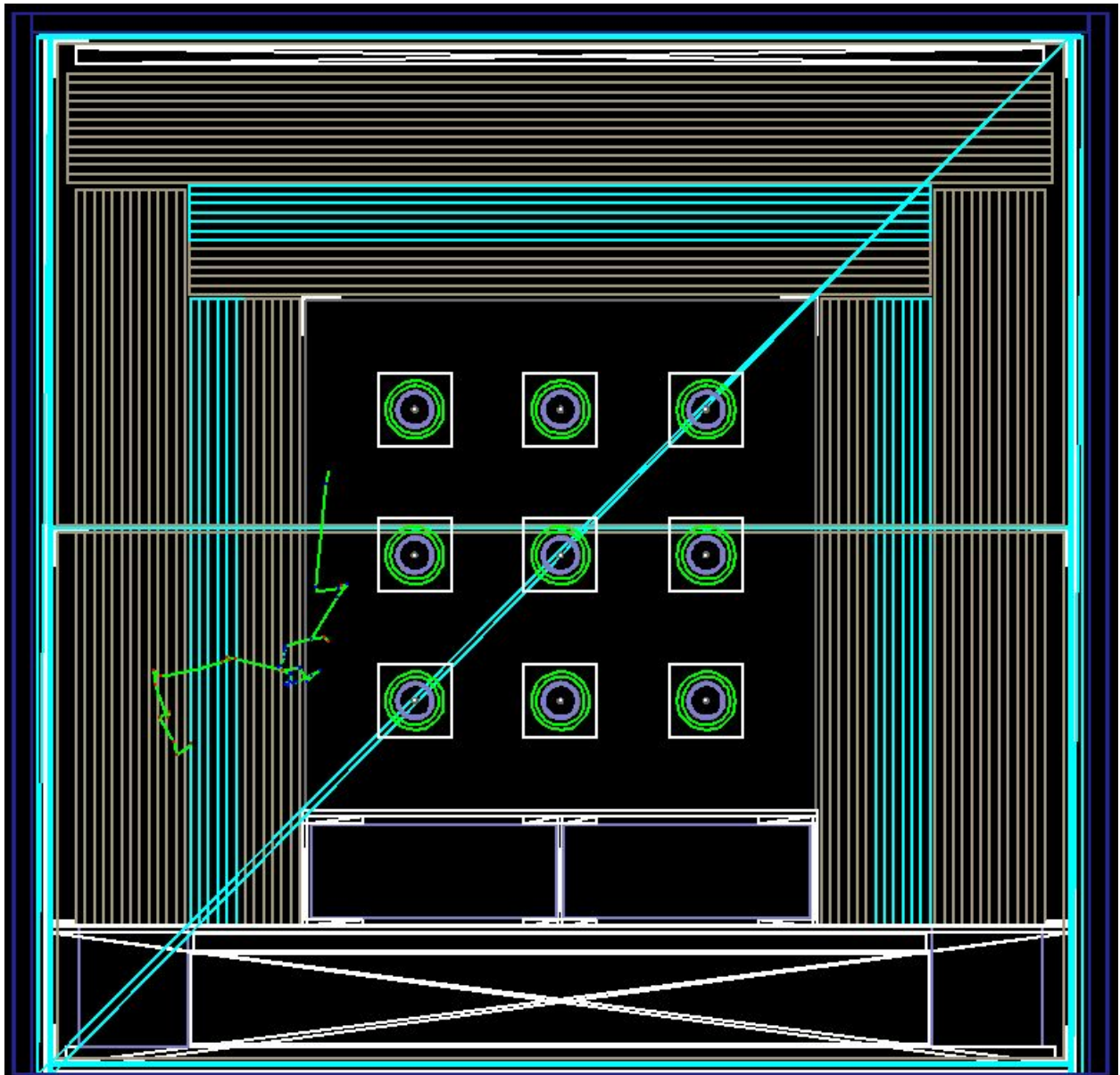
```
/miniHALO/DetectorConstruction/HDPEMaterials Air
```



The diagrams showing the differences to the detector as a result of these parameters can be compared to the last image in section 2.5, which is the same view of the detector without any `/miniHALO/DetectorConstruction/` parameters specified.

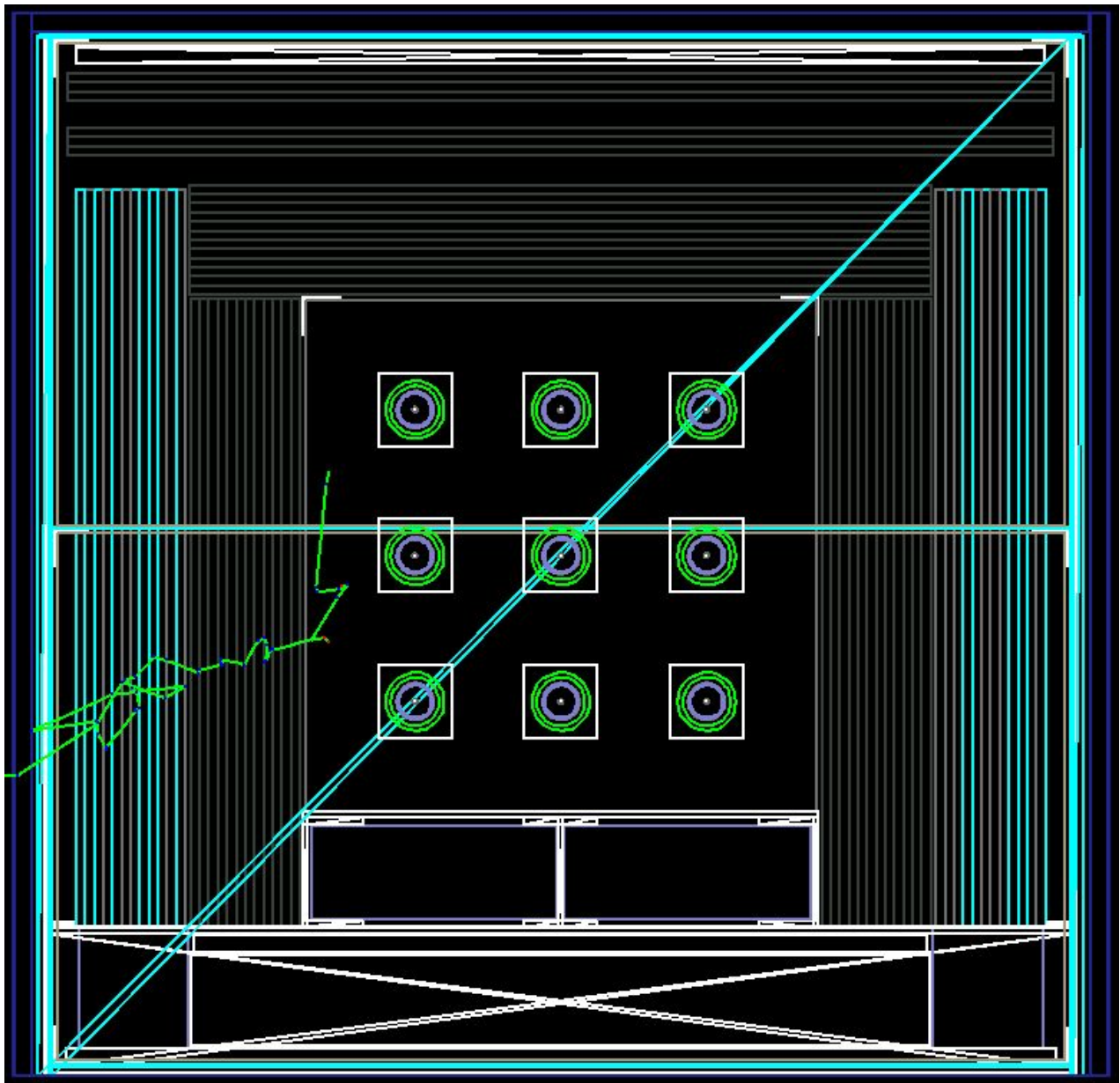
Example #2: Half graphite to HDPE, half to Aluminum. We can also specify “all” as the side, which is functionally exactly the same as not specifying a side. Specifying two materials changes the innermost half of the layers into the first material and the outermost half of the layers into the second material specified.

```
/miniHALO/DetectorConstruction/GraphiteMaterials all-HDPE-Aluminum
```



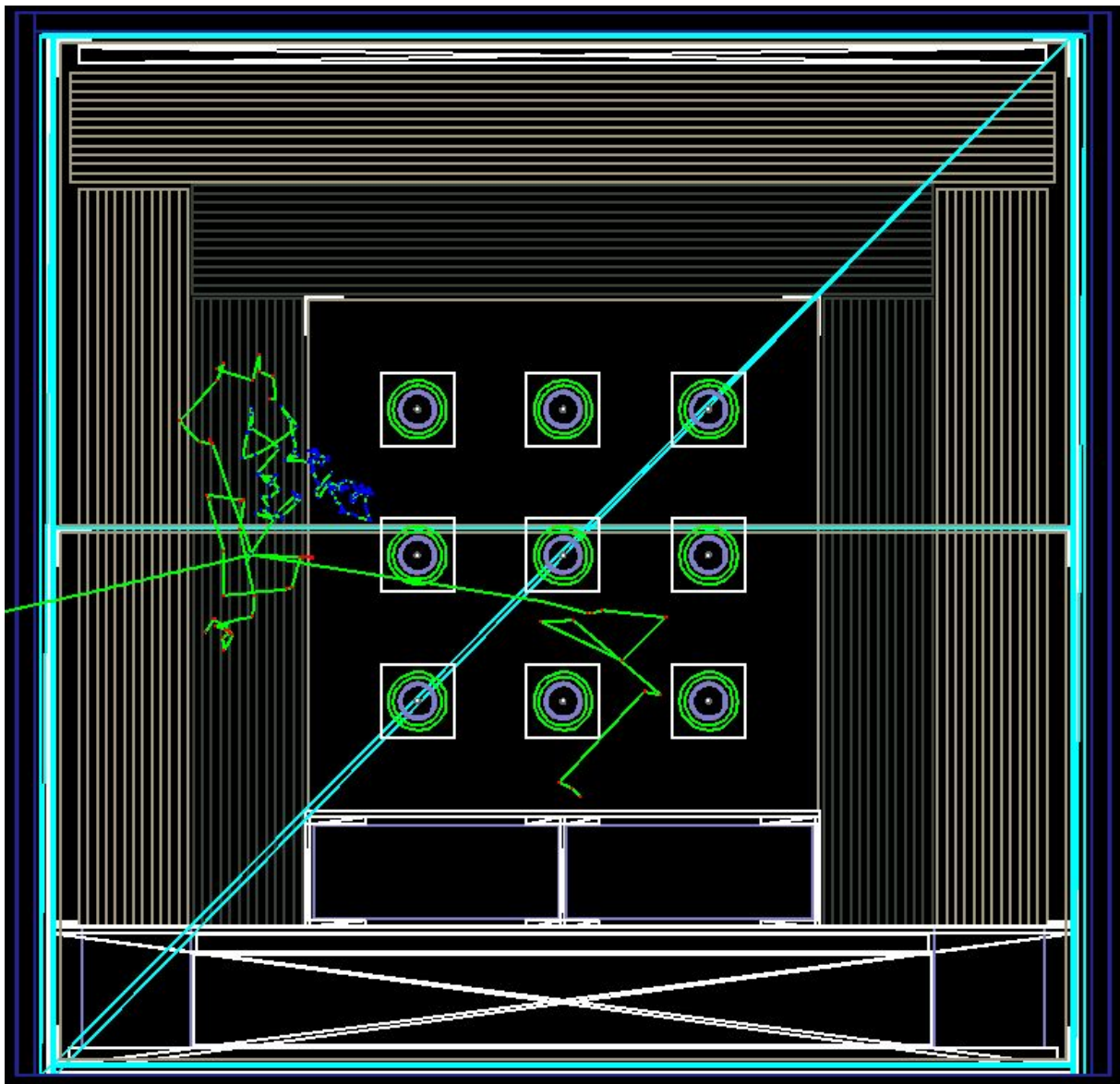
Example #3: Top HDPE to alternating 1.5 inch layers of air and graphite, right and left HDPE to alternating 0.5 inch layers of lead and aluminum. This example shows off more ways to slice the layers (in 4 segments and 12 segments in these examples) and shows the same parameter being called multiple times to set different sides of the detector with different material configurations. To be clear, the screenshot of the parameter call is word-wrapping in the middle of the word “Aluminum”, it is not broken into three lines.

```
/miniHALO/DetectorConstruction/HDPEMaterials top-Air-graphite-Air-graphite  
/miniHALO/DetectorConstruction/HDPEMaterials left-right-Lead-Aluminum-Lead-Aluminum-Lead-Alu  
minum-Lead-Aluminum-Lead-Aluminum-Lead-Aluminum
```



The LeadMaterials parameter is more simple than the other two material parameters due to the fact that the lead block is all one volume without layers or sides, and therefore the parameter always accepts only a single material name. The example below, of this parameter, shows the change to a block of HDPE in place of the lead block.

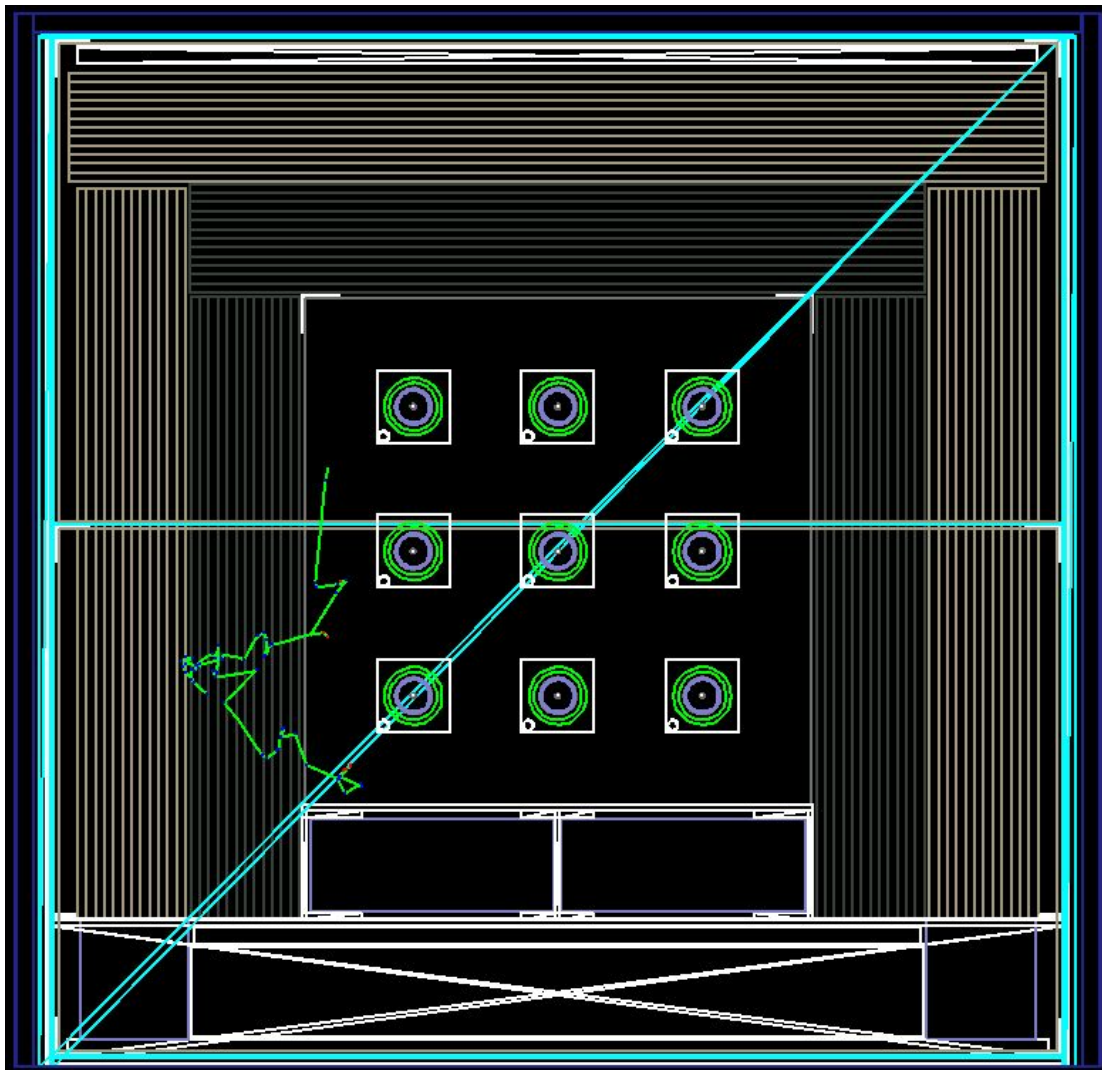
```
/miniHALO/DetectorConstruction/LeadMaterials HDPE
```



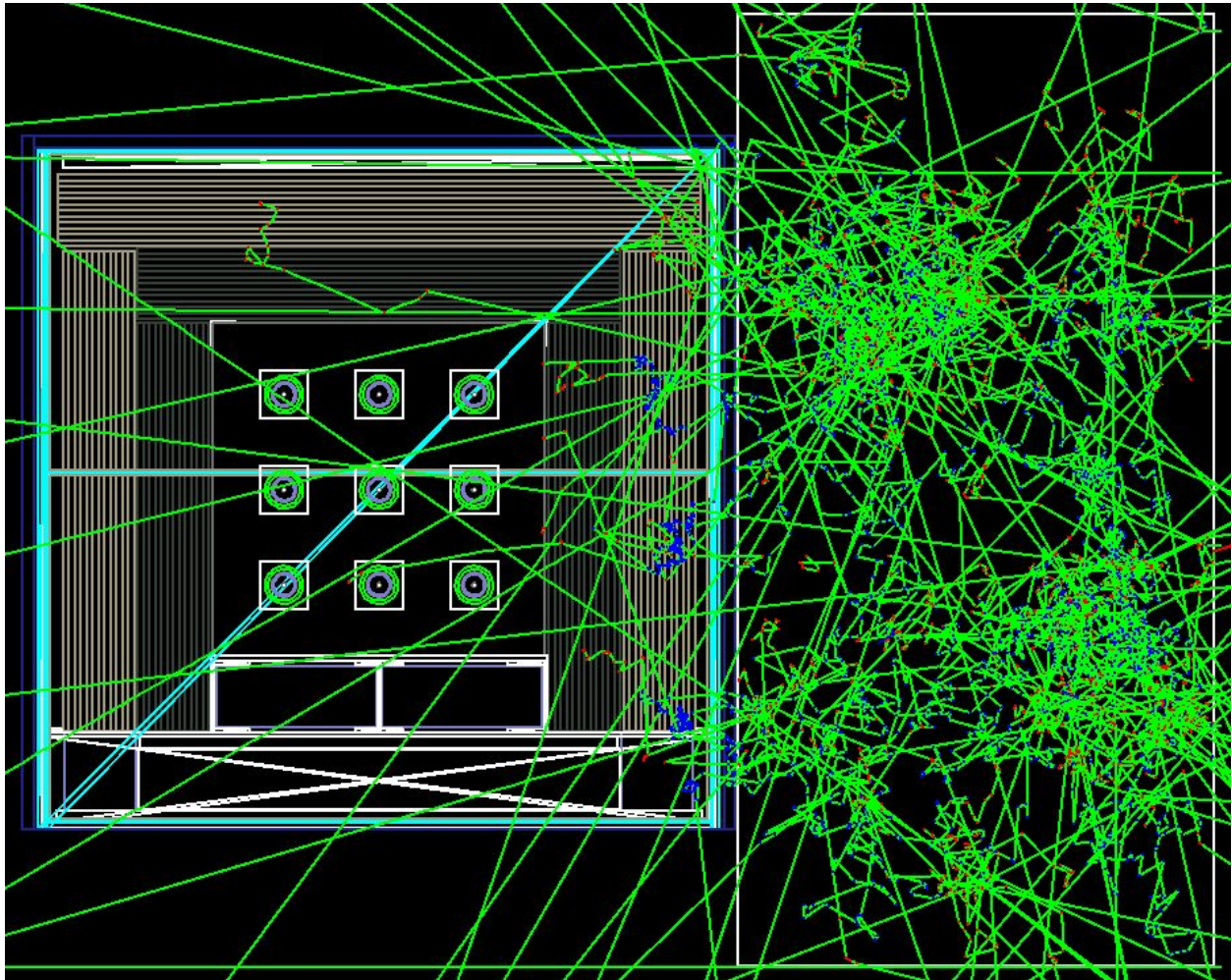
3.3 IncludeConcreteShield / IncludeCalibrationTubes

These two binary parameters take a true or false value (I always used 0 for false and 1 for true, but other values that can be evaluated to true or false will also work). If the value is true, that part of the detector is instantiated when the detector is constructed. If false, that section of the detector is not created. The purpose of these parameters is primarily so that the macros for the different neutron generation modes (Cf-252 calibration, break-through neutrons, break-through-with-shielding) can enable those aspects of the detector that are relevant to them. For the “default” situation of having neutrons generated within the lead volume both the calibration tubes and concrete block are not present. The following screenshots show the results of enabling the parameters.

```
/miniHALO/DetectorConstruction/IncludeCalibrationTubes 1
```




```
/miniHALO/DetectorConstruction/IncludeConcreteShield 1
```



The above screenshot of the detector with concrete shielding includes 10 neutrons instead of the single neutron being simulated in all the other screenshots so far, and because the break-through neutrons are so high-energy compared to the Cf-252 neutron energy spectrum used, there are a lot of particle tracks being visualised here (the green lines), but the actual concrete shielding is the white box to the right of the detector.

3.4 CheckOverlaps

The CheckOverlaps parameter is the only one I've implemented that doesn't actually affect the geometry of the detector. When creating new volumes, set it to true in order to have every volume being created check if it's creation will overlap with any existing volumes.

```
/miniHALO/DetectorConstruction/CheckOverlaps 1
```

The way it works is that when creating physical volumes you can set a boolean parameter to check for overlaps on creation and all the volumes created so far have that boolean set as the parameter value (which defaults to false). The reason I'm mentioning it is because if you are creating new volumes you should follow the same pattern I've established in DetectorConstruction and make sure that the `check_overlaps` variable is set in the last position of the physical volume initialization.

```
G4VPhysicalVolume* p_aluminium_endcap_back = new G4PVPlacement(0, G4ThreeVector(0 * inch, 0 * inch, -40.8 * inch), l_aluminium_endcap, "p_aluminium_endcap_back", l_world, false, 0, check_overlaps);
G4VPhysicalVolume* p_aluminium_endcap_front = new G4PVPlacement(0, G4ThreeVector(0 * inch, 0 * inch, 53.1875 * inch), l_aluminium_endcap, "p_aluminium_endcap_front", l_world, false, 0, check_overlaps);
```

The result of turning this parameter on or off is that the text output of running a simulation will include or exclude a block of text that will be very similar to this:

```
Checking overlaps for volume p_lead_box (G4Box) ... OK!
Checking overlaps for volume p_hole_in_lead_0_0 (G4Box) ... OK!
Checking overlaps for volume p_NCD_0_0 (G4Tubs) ... OK!
Checking overlaps for volume p_detector_tube_0_0 (G4Tubs) ... OK!
Checking overlaps for volume p_hole_in_lead_0_1 (G4Box) ... OK!
Checking overlaps for volume p_NCD_0_1 (G4Tubs) ... OK!
Checking overlaps for volume p_detector_tube_0_1 (G4Tubs) ... OK!
Checking overlaps for volume p_hole_in_lead_0_2 (G4Box) ... OK!
Checking overlaps for volume p_NCD_0_2 (G4Tubs) ... OK!
Checking overlaps for volume p_detector_tube_0_2 (G4Tubs) ... OK!
Checking overlaps for volume p_hole_in_lead_1_0 (G4Box) ... OK!
Checking overlaps for volume p_NCD_1_0 (G4Tubs) ... OK!
Checking overlaps for volume p_detector_tube_1_0 (G4Tubs) ... OK!
Checking overlaps for volume p_hole_in_lead_1_1 (G4Box) ... OK!
Checking overlaps for volume p_NCD_1_1 (G4Tubs) ... OK!
```


There will be as many lines as there are volumes in the simulation. If any of these lines say “Overlap is detected!” instead of “OK!”, then two volumes overlap. This will often crash a simulation because if any particles are simulated in the overlapping space they won’t know which material they are in and the simulation will break down. To avoid that, you’ll need to resize or reposition one or both of the volumes to match the configuration of the detector.

Protip: If volumes are bounded by the exact same coordinates, they are treated as overlapping. Consider spacing your volumes a tiny bit even if they are supposed to be adjacent.

4 Changing the Code

The code repository is currently on Nearline at the following location:

```
cd /project/halo/mini_halo/miniHALO/
```

It can also be found on GitHub, as of August 2020 it can be located at <https://github.com/RemiHill/miniHALO>, though I believe it’s access is by invite only. If all you’re doing is modifying the aspects of the detector that can be accessed via parameters (section 3), use them instead of changing the code.

4.1 Project layout

The miniHALO directory contains a number of files in addition to two subdirectories. The files are primarily makefiles and macros for running the simulation. The `kaelan_analysis.cc` file is the post-simulation ROOT file I put together, explained in more detail in section 5. The actual code that makes up the simulation is located in the `/src/` and `/include/` subdirectories. The former contains all the source code, the latter all the header files.

In the `src` directory there are 9 files, only a few of which I ever really worked on. `miniHALOAnalysis.cc` contains all the code that Remington wrote to set up the TTree values for the ROOT analysis, `miniHALOPrimaryGeneratorAction.cc` contains all the code that Remington wrote to set up using the Geant4 General Particle Source to manage particle creation using the macro commands discussed in section 2.2.

The two files I really worked with in the src directory are the `miniHALODetectorConstruction.cc` and the `miniHALODetectorMessenger.cc` files. The former contains most of the code that establishes the detector's geometry; the latter contains the code that manages the parameters discussed in section 3. Instead of going through how the code works in those two files line-by-line, I'm going to go through what I consider to be probable things you might want to do when changing the code and explain, in broad strokes, how to do them.

Case #1: Adding a parameter

In order to add a brand new parameter to the simulation, you'll need to follow a few steps in all four files.

`miniHALODetectorMessenger.hh`

- You'll need to add a `G4UICmd` declaration using the same format as found for other parameters in that file.

`miniHALODetectorMessenger.cc`

- In the constructor you'll need to copy/paste and modify the four lines that initialize the `G4UICmd` object declared in the previous step. All these are pretty self-explanatory, the only thing that usually changes is the explanation in the `SetGuidance` command.
- Add a delete line in the destructor.
- In the `SetNewValue` function add a new if branch that calls the `Set` function you'll create in the next files, to the new value converted to the appropriate type to your new parameter.

`miniHALODetectorConstruction.hh`

- Add a `Set` function declaration as a public member with appropriate arguments.
- Add one or more variables that will drive the purpose of your parameter as private members. Make sure to comment the purpose of the variable, which can probably be copy/pasted from `SetGuidance` in the previous file.
- Add a `Get` function declaration as a private member.

`miniHALODetectorConstruction.cc`

- Specify the default value of the parameter in the constructor using the `Set` function.
- Define the `Set` function of the parameter.
- Define the `Get` function of the parameter.
- Use the variables being set to modify the `Construct` function to suit your needs.

Case #2: Adding a material

`miniHALODetectorConstruction.hh`

- Ensure the constituent materials/elements are declared as private members.
- Declare the new material as a private member.

`miniHALODetectorConstruction.cc::SpecifyMaterials()`

- Ensure all the constituent elements and materials are defined. If they aren't, define them in the same way that others are defined.
- Define your new material by specifying the density, name, and number of component elements/materials. Assign the components to the new material.

Protip: The only material I had to define is Aluminum6063, so check that out for an example of the above material definition.

Case #3:

4.2 Project standards

The purpose of this section is to try and maintain some consistency and readability in the project's code, especially due to the cleanup that Remington Hill and I had to perform as a result of people using their own styles and formats in the project prior to the summer of 2020. These "best practices" might be out of date but unless directed by your supervisor or a more recent document explaining project standards, I advise that anyone modifying the Mini-HALO Monte Carlo simulation code try to adhere to these guidelines.

Variables should be in all lowercase letters, with words being separated by underscores ('_'). In any instance where a Geant4 volume is being defined, the solid / logic / physical volume should start with `s_ / l_ / p_` and the volume name should match the G4 object name (variable name matches the first string parameter). Variable names should be as self-descriptive as possible not only to assist in code readability but also to assist in debugging issues when the G4 object name is available. Ideally variable names should not include numbers (i.e. `s_angle_1`, `s_angle_2`, etc.) since this isn't descriptive at all but some exceptions apply (like the 12 layers of the left graphite block).

Comments should be plentiful in the code, explaining what the reasoning behind a certain block of code or as an overview of a function. Without comments, the next person who modifies the project will have a much harder time following the reasoning

behind your changes and might overwrite them since they don't understand the purpose of your changes.

As of August 2020, most simulations will be run on SNOLAB's high-performance computing cluster "Nearline". That makes efficiency more important than ever because if we are negligent in ensuring that the simulations are as fast as possible, we are needlessly holding up computing nodes that could be used by other researchers. To that effect, try to optimize the code. Avoid re-initializing an identical object within a loop (like most physical and logical volumes), avoid branching when possible, avoid using expensive operations when a simpler one will do (i.e. $A * A$ instead of A^2); and avoid evaluating the same thing over and over again (i.e. same x dimension of many volumes). I tried to clean up the `DetectorConstruction` function before writing this user manual, but there is work that can be done in other functions and files if you're finding that simulations are running more slowly than expected.

5 ROOT Analysis

The ROOT analysis was facilitated largely by Remington Hill, though I will provide the insights into the process that I can to facilitate use. We've implemented 27 one-dimensional histograms and two-dimensional scatter plots to show different aspects of the data collected during the simulation. That data is collected and stored by objects in the `/project/halo/mini_halo/miniHALO/src/miniHALOAnalysis.cc` file. The graphs are put together by ROOT, utilizing the C++ commands that make up the `/project/halo/mini_halo/miniHALO/kaelan_analysis.cc` file. To create new graphs, follow the examples in `kaelan_analysis.cc`, unfortunately I can't guide you in manipulating the `miniHALOAnalysis.cc` file.

5.1 Accessing implemented ROOT graphs

To see the graphs that have already been put together, you'll need to first run a simulation with as many particles as possible (10,000 neutrons has been my go-to for a quick simulation that has enough data to actually see what's going on). Once you've completed the run, move to the following directory:

```
cd /project/halo/mini_halo/miniHALO-build/
```

Then launch ROOT, the `-l` launches without the logo and passing the analysis file automatically runs that code when ROOT starts.

```
root -l ../miniHALO/kaelan_analysis.cc
```

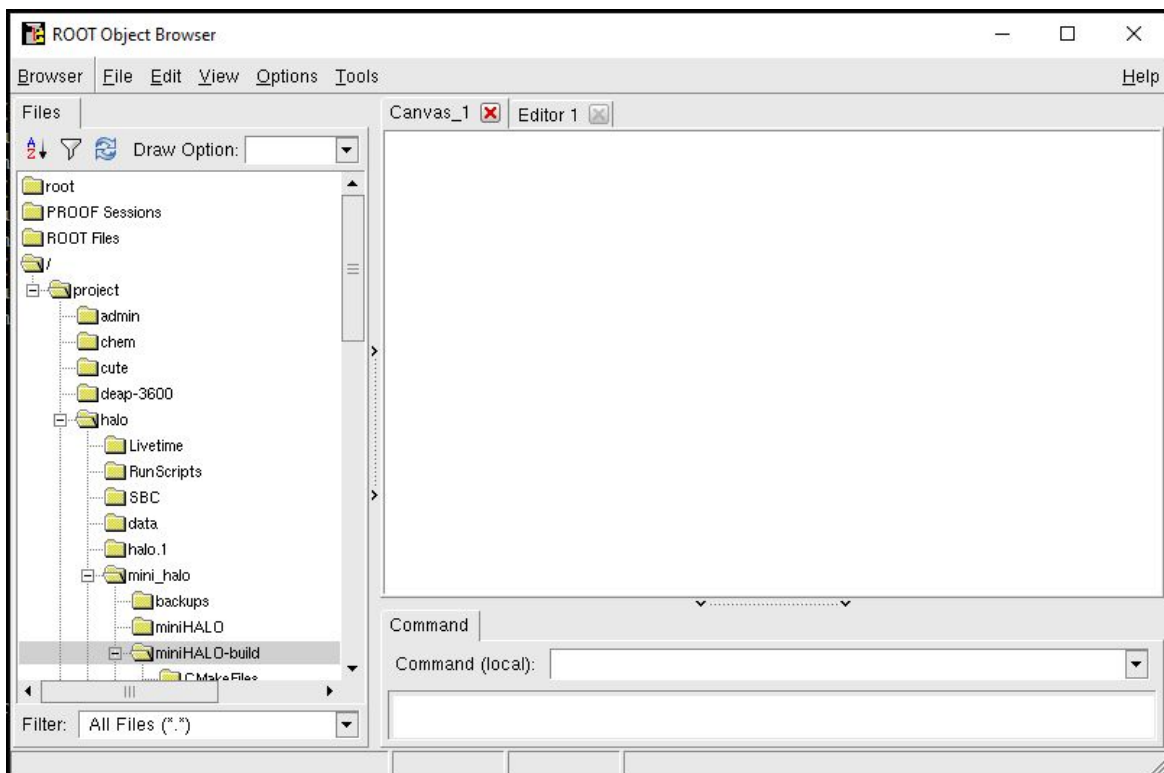
When you'll need to wait for the above commands to complete, which will look similar to the following screenshot:

```
[krenault@nearline-login miniHALO-build]$ root -l ../miniHALO/kaelan_analysis.cc
root [0]
Processing ../miniHALO/kaelan_analysis.cc...
root [1] █
```

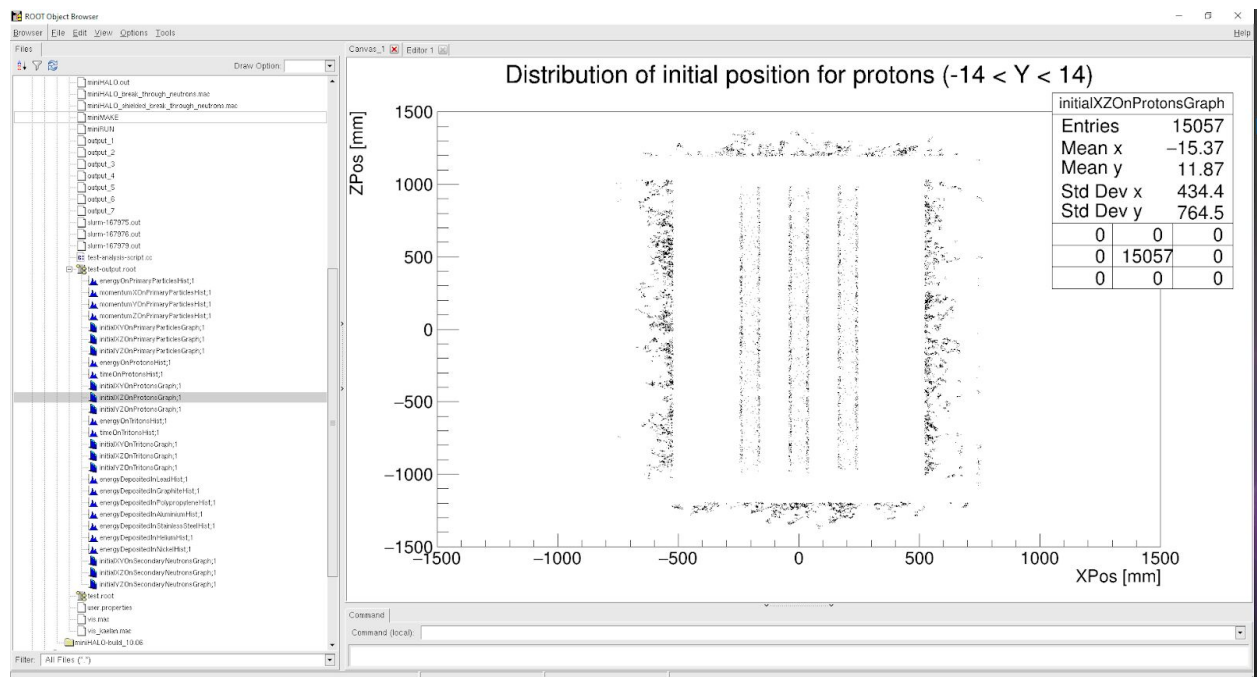
Once “root [1]” is visible, you can launch the TBrowser to view the histograms created. I used the variable name “b”, but it’s completely arbitrary.

```
[krenault@nearline-login miniHALO-build]$ root -l ../miniHALO/kaelan_analysis.cc
root [0]
Processing ../miniHALO/kaelan_analysis.cc...
root [1] TBrowser b
```

The window that appears should look similar to the following:



Scroll down in the menu on the left until you see test-output.root, then click on it. You should now be able to click on any of the graphs that are nested in that section and see the data. See the following screenshot for an example.



This last screenshot is a zoomed-in version of the menu on the left of the above image.

