Simulation of In-Ice Radio Emission of Cosmic Ray Air Shower Cores using Geant4

Simon De Kockere

February 14, 2024

1 Introduction

In this document I will explain the workflow that needs to be followed in order to run a full simulation, combining both the in-air and in-ice radio emission for in-ice radio antennas coming from a cosmic ray air shower.

All the scripts needed to run and process simulations can be found in */software/geant4/air_and_ice_comb/submit_scripts* on the t2b cluster. Copy this folder to your local workspace, to avoid editing the original scripts.

2 Running Corsika with CoREAS

To run the in-air simulation part, you will need to install Corsika with the modified version of CoREAS. Once installed, you can use the script *run_corsika.sh* to run a Corsika simulation, which is can be submitted to the cluster using the *run_corsika.submit* file. Before you can use these two scripts, you will have to:

1. make sure that the variable *executable* in *run_corsika.submit* points to your *run_corsika.sh* script.
2. make sure that the variable *error* in *run_corsika.submit* points to the directory where you want to store the error files.

Now you can submit a job to the cluster using the submit file. See the submit file itself for more information on how to use it for submitting a job. The submit file needs 6 arguments:

- *INPUT_FILE*: the Corsika steering file, which lists the run number, event number, different seeds, energy cut values etc.
- *RUN_NR*: the 6-digit run number that is also specified in the Corsika steering file, e.g. 000001
- *ATMOS_FILE*: the *atmosphere.dat* file defining the atmospheric properties
- *REAS_FILE*: the *SIM.reas* file which will be used by CoREAS. See the CoREAS manual for more information. Note that CoREAS expects a file of the name "SIMXXXXXX.reas", with XXXXXX the run number. However, the file provided to this submit script **should be of the more simpler form "SIM.reas"**. The run number will automatically be added to a local copy of the reas-file on the cluster node.
- *LIST_FILE*: the *SIM.list* file which will be used by CoREAS. Note that CoREAS expects a file of the name "SIMXXXXXX.list", with XXXXXX the run number. However, the file provided to this submit script **should be of the more simpler form "SIM.list"**. The run number will automatically be added to a local copy of the list-file on the cluster node.
- *OUTPUT_DIR*: the output directory to write the simulation output files to
- *LOG_NAME*: a string identifier for the job that will be added to the error file name

After the job finished, you might have an error in the error output file stating something like "*Note: The following floating-point exceptions are signalling: IEEE_INVALID_FLAG IEEE_UNDERFLOW_FLAG IEEE_DENORMAL*", which you can just ignore...

3 Preparing the input files for Geant4

Next, you will have to prepare the input files for Geant4, using the Corsika output file (DATXXXXXX, with XXXXXX the run number). For this, use the *make_job_files.sh* and *make_job_files.submit* scripts. It will read the Corsika particle output file, select those particles that fall within a given footprint on the ice and prepare input files for the Geant4 simulation. Before you can use these two scripts, you will have to:

1. make sure that the *corsikaread_thin* variable in *make_job_files.sh* points to the corsikaread_thin (or corsikaread) executable. This executable will translate the DATXXXXXX binary file to a human readable file, which will then be used by some python scripts to generate input files for the Geant4 simulation. You can find the script in the src/utils/ directory of your Corsika instalation. This is a fortran code which needs to be compiled in order to generate the executable. Instructions on how to compile this code can be found at the top of the fortran script itself.

   IMPORTANT: Before you compile this script, you will need to change a single line, which will increase the precision with which it writes to the output file. On line 122 you will find

   WRITE(8,'(1P,E16.8,7E13.5)') (PDATA(II+IL),II=0,7)

   which needs to be changed to

   WRITE(8,'(1P,E16.8,7E16.8)') (PDATA(II+IL),II=0,7)

2. make sure that the *upper_radius* variable in *make_job_files.sh* has the correct value. It represents the radius of the air shower footprint that you want to propagate through the ice, **in cm**. It is defined in the plane perpendicular to the shower, i.e. the actual footprint will be an ellipse.

3. make sure that the variable *executable* in *make_job_files.submit* points to your *make_job_files.sh* script.

4. make sure that the variable *error* in *make_job_files.submit* points to the directory where you want to store the error files.

Now you can submit a job to the cluster using the submit file, to generate input files for the Geant4 simulation. See the submit file itself for more information on how to use it for submitting a job. The submit file needs 5 arguments:

- *INPUT_FILE*: the output file generated by Corsika, of the form DATXXXXXX
- *OUTPUT_DIR*: the output directory to store the Geant4 input files in.
- *ZENITH*: the value of the zenith angle from the Corsika simulation, **in degrees**
- *AZIMUTH*: the value of the azimuth angle from the Corsika simulation, **in degrees**
- *CORSIKA_LOG_FILE*: the log file generated by the Corsika simulation, of the form RUNXXXXXX.log
- *LOG_NAME*: a string identifier for the job that will be added to the error file name

After the job finished, you might have errors in the error output file stating something like "*Error in <TSystem::ExpandFileName>: input: $HOME/0, output: $HOME/0*", which you can just ignore...

4 Running the Geant4 simulation

Now you have a directory filled with text files that each serve as the input for a single Geant4 job. Each file has the same header, which contains information about the file (e.g. what the numbers mean, and which unit system we're using), followed by lines of numbers. Each of these lines represents a different particle from the air shower footprint, and contains the information that will be given to the Geant4 simulation.

In principle it would have been possible to list all the particles in a single file, and use this file as the input for the Geant4 simulation. However that would take weeks, if not months, to complete the Geant4 simulation. To speed up the process we've split up this single file into multiple files, so we can run the simulation on different nodes at the time, simply by giving each note a different, smaller input file. Splitting up the single input file is done based on the energy of the particles. Input files listing low-energy particles typically contain much more particles for a single job compared to input files for high-energy particles. This way we don't overwhelm the cluster with thousands of jobs, while still keeping total run time under control. The name of each text file indicates the energy range of the particles it has listed.

We will now have to submit a single job for every input file in the directory. Luckily, there's an automated system to do that for us, which is steered through a dag-file. A dag-file is a list of all the jobs you want to submit to the cluster, which can then be submitted to the cluster as a single job. The dagman will then automatically handle the submission of the jobs for you. To make a dag-file, use the *make_dagfile.py* script (which was written for python2, but should also work fine with python3). Before you can use this script, you will have to:

1. make sure that the variable *ice_shelf_exe* in the *run_ice_shelf_project.sh* script points to the correct Geant4 *ice_shelf* executable (i.e. the executable for the Geant4 simulation). By default, it will be set to point to the executable in */software/geant4/air_and_ice_comb*.
2. make sure that the *run_ice_shelf_project.sh* script sources the correct *geant4.sh* script, to set up an available Geant4 installation
3. make sure that the variable *executable* in *run_ice_shelf_project.submit* file points to your *run_ice_shelf_project.sh* script.
4. make sure that the variable *error* in *run_ice_shelf_project.submit* points to the directory where you want to store the error files. Do **not** change the error file names themselves (*$(LOG_NAME)_CoREAS.err*), as this specific name will be expected by the script described in the next paragraph.
5. make sure that the variable *submit_file* in the script points to your *run_ice_shelf_project.submit* file.

Now you can create the dag-file for the Geant4 simulation. The python script needs 9 arguments, in the following order:

- the input directory containing all the input files for the simulation
- the output directory to store all the output files of the simulation, which should NOT be the same as the input directory mentioned above
- a single string identifying your simulation, which will be added to the error files;
  e.g. “runnr_primary-energy_zenith_azimuth”
- the output directory to write the dag-file to
- the zenith angle of the air shower, **in degrees**
- the azimuth angle of the air shower, **in degrees**
- the *SIM.reas* file used for the Corsika simulation
- the *SIM.list* file used for the Corsika simulation
- the *atmosphere.dat* file used for the Corsika simulation

With that done, you now have a dag-file that you can submit to the cluster. To do so, simply go to the directory that has the dag-file, and use *condor_submit_dag name-of-dag-file*.

5 Verifying the Geant4 simulation

If one of the jobs managed by the dagman on the cluster did not finish correctly (i.e. gave an error message), the dagman should have created a “rescue” file in the same directory as the dag-file. This “rescue” file lists the failed jobs, and will make sure that by simply resubmitting the dag-file again, the dagman will only submit those jobs that failed.

However, I noticed that this automated “rescue” system does not work in our case, for which I have not found the reason yet. However, I implemented my own “rescue” system using the python3 script *check_for_killed_jobs.py*. This script expects one argument, which is the dag-file you used for running the Geant4 simulation.

This script will check the error files generated during the simulation, and will create a new dag-file listing all the jobs that have an error stating the corresponding job was killed. Usually this means something random went wrong, and you can just submit the new dag-file to get these randomly killed jobs running again.

The script will also notify you of jobs with any other type of error message, but it will **not** automatically include these jobs in the new dag-file. Other error messages should be investigated first before resubmitting their corresponding jobs, although I found that most errors simply solve themselves by resubmitting the failed jobs, which you can do by manually adding these jobs to the new dag-file. The script will print the *VARS* lines of these jobs to the terminal which you can simply copy, but don’t forget to add the *JOB* line yourself.

6 Combining the Geant4 output files

Every single job will have created its own output files in the given directory. There will be *root* files, which contain some information about the in-ice particle cascade, and text files, which have the antenna trace information. The final step is to combine all these files. Combining the root files can be done using the *run_combine_root_files.sh* and *run_combine_root_files.submit* scripts. Before you can use these two scripts, you will have to:

1. make sure that the variable *executable* in *run_combine_root_files.submit* points to your *run_combine_root_files.sh* script.
2. make sure that the variable *error* in *run_combine_root_files.submit* points to the directory where you want to store the error files.

Combining the text files can be done using the *run_combine_antenna_traces.sh* and *run_combine_antenna_traces.submit* scripts. Before you can use these two scripts, you will have to:

1. make sure that the variable *executable* in *run_combine_antenna_traces.submit* points to your *run_combine_antenna_traces.sh* script.
2. make sure that the variable *error* in *run_combine_antenna_traces.submit* points to the directory where you want to store the error files.

Now you can submit a job for combining the root files, and a job for combining the text files to the cluster. See the submit files for more information on how to use them for submitting a job. Both submit files need 2 arguments:

- *INPUT_DIR*: the directory holing all the root/text files that need to be combined
- *LOG_NAME*: a string identifier for the job that will be added to the error file name

Once the job is done, you should be able to find the following files in the same directory that holds all the Geant4 output files you combined:

- *histos_combined.root*: information about the in-ice cascade
- *antenna_traces_comb.txt*: electric field in function of time for every antenna in the simulation
- *antenna_traces_comb_dir.txt*: electric field in function of time for every antenna in the simulation, **direct emission only**
- *antenna_traces_comb_indir.txt*: electric field in function of time for every antenna in the simulation, **indirect emission only**
- *antenna_traces_comb_SA.txt*: electric field in function of time for every antenna in the simulation, **sudden appearance emission only (both direct and indirect emission)**

Unit of time and electric field are noted down in the text files themselves. **The coordinate system used is that of Geant4!**

Once you made sure the combined files are there and are healthy, you can remove the seperate output files created by the different jobs, simply by going into the directory containing the output files and using the command *rm geant4**

Just as a side note, the files *antenna_traces_comb_dir.txt* and *antenna_traces_comb_indir.txt* also contain the contribution from the sudden appearance, so you have:

antenna_traces_comb.txt = *antenna_traces_comb_dir.txt* + *antenna_traces_comb_indir.txt*

and **NOT**

antenna_traces_comb.txt = *antenna_traces_comb_dir.txt* + *antenna_traces_comb_indir.txt* + *antenna_traces_comb_SA.txt*,

which would count the sudden appearance emission twice.

7 Coordinate systems

An overview of the Corsika/CoREAS axis system versus the Geant4 axis system can be seen in Figure 1. To go from Corsika/CoREAS spatial coordinates to Geant4 spatial coordinates, we use:

$$
\begin{aligned}
x_G &= x_C \cos(\phi) + y_C \sin(\phi) \\
y_G &= Z_C + d \\
z_G &= -(y_C \cos(\phi) - x_C sin(\phi)),
\end{aligned}
$$

which also holds for momentum and electric field if omitting the spatial translation ("$+d$") for the y-component. To go from Geant4 spatial coordinates to Corsika/CoREAS spatial coordinates, we use:

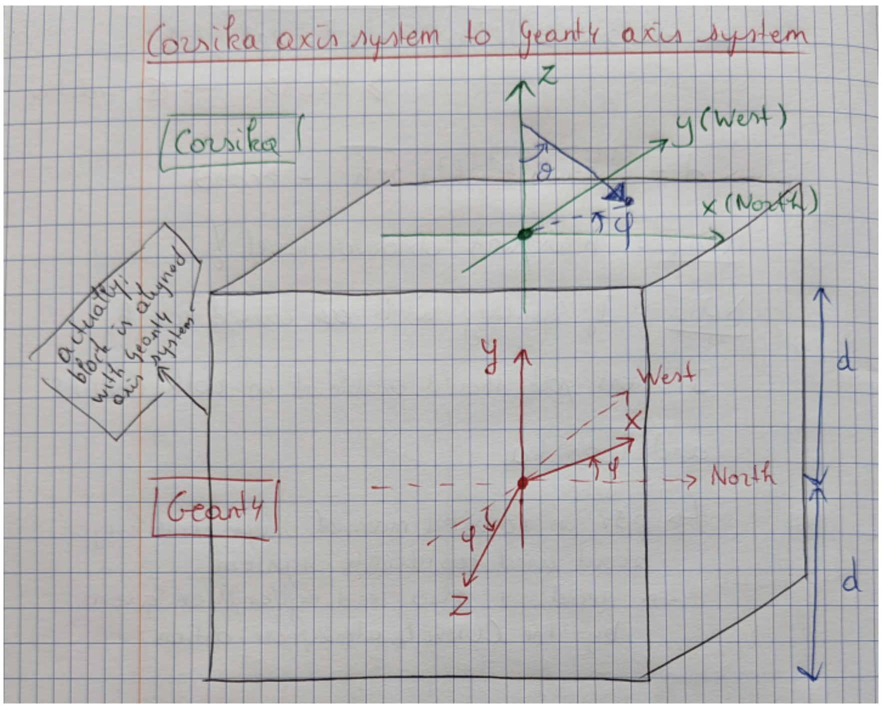
$$
\begin{aligned}
x_C &= x_G \cos(\phi) + z_G \sin(\phi) \\
y_C &= x_G \sin(\phi) - z_G \cos(\phi) \\
z_C &= y_G - d,
\end{aligned}
$$

again omitting the spatial translation for the z-component in case of momentum and electric field.

I provided a python script which converts the in-ice traces from the Geant4 axis system to the Corsika/CoREAS axis system as well as move to the unit system used by CoREAS (CGS-units), called *g4_to_CoREAS.py*. It needs 3 arguments, in the following order:

- The in-ice trace file created by the Geant4 simulations.
- The output directory to store the converted trace files in (one per antenna).
- The azimuth angle ϕ of the shower that was simulated, **in degrees**.

You will get a short message telling you to what precision the time values will be stored in the new output file (set to 1e-6 ns per default), since the script will round up/down the time values of the traces to avoid values like 1.23e-7 ns to become 1.230000000001e-7 ns. In case you need better precision, you can adjust the *tround* parameter in the script.



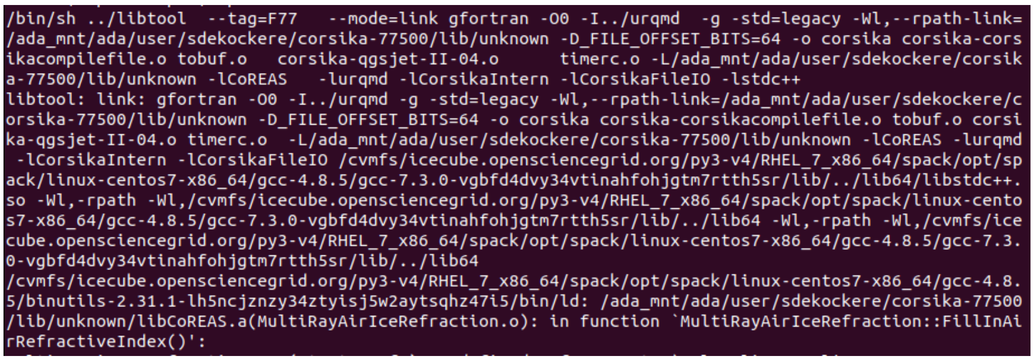
Corsika axis system to Geant4 axis system
Corsika
Z
y (West)
x (North)
θ
φ
actually: block is aligned with Geant4 axis system
y
West
X
North
φ
Geant4
φ
Z
d
d

Figure 1: The Corsika/CoREAS axis system versus the Geant4 axis system.

APPENDIX A: installing Corsika with modified CoREAS

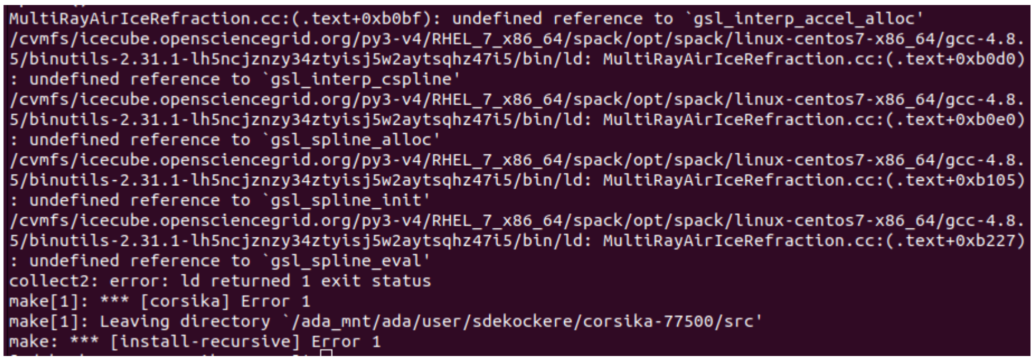
To install Corsika with the modified CoREAS version, simply use the *coconut* script provided with the source code, just like you would when installing with the 'normal' version of CoREAS.

Unfortunately, the *coconut* script will not be able to complete the full installation, and will exit giving you an error. This should look something like:



```
/bin/sh ../libtool  --tag=F77   --mode=link gfortran -O0 -I../urqmd  -g -std=legacy -Wl,--rpath-link=
/ada_mnt/ada/user/sdekockere/corsika-77500/lib/unknown -D_FILE_OFFSET_BITS=64 -o corsika corsika-cors
ikacompilefile.o tobuf.o   corsika-qgsjet-II-04.o      timerc.o -L/ada_mnt/ada/user/sdekockere/corsik
a-77500/lib/unknown -lCoREAS   -lurqmd -lCorsikaIntern -lCorsikaFileIO -lstdc++
libtool: link: gfortran -O0 -I../urqmd -g -std=legacy -Wl,--rpath-link=/ada_mnt/ada/user/sdekockere/c
orsika-77500/lib/unknown -D_FILE_OFFSET_BITS=64 -o corsika corsika-corsikacompilefile.o tobuf.o corsi
ka-qgsjet-II-04.o timerc.o  -L/ada_mnt/ada/user/sdekockere/corsika-77500/lib/unknown -lCoREAS -lurqmd
 -lCorsikaIntern -lCorsikaFileIO /cvmfs/icecube.opensciencegrid.org/py3-v4/RHEL_7_x86_64/spack/opt/sp
ack/linux-centos7-x86_64/gcc-4.8.5/gcc-7.3.0-vgbfd4dvy34vtinahfohjgtm7rtth5sr/lib/../lib64/libstdc++.
so -Wl,-rpath -Wl,/cvmfs/icecube.opensciencegrid.org/py3-v4/RHEL_7_x86_64/spack/opt/spack/linux-cento
s7-x86_64/gcc-4.8.5/gcc-7.3.0-vgbfd4dvy34vtinahfohjgtm7rtth5sr/lib/../lib64 -Wl,-rpath -Wl,/cvmfs/ice
cube.opensciencegrid.org/py3-v4/RHEL_7_x86_64/spack/opt/spack/linux-centos7-x86_64/gcc-4.8.5/gcc-7.3.
0-vgbfd4dvy34vtinahfohjgtm7rtth5sr/lib/../lib64
/cvmfs/icecube.opensciencegrid.org/py3-v4/RHEL_7_x86_64/spack/opt/spack/linux-centos7-x86_64/gcc-4.8.
5/binutils-2.31.1-lh5ncjznzy34ztyisj5w2aytsqhz47i5/bin/ld: /ada_mnt/ada/user/sdekockere/corsika-77500
/lib/unknown/libCoREAS.a(MultiRayAirIceRefraction.o): in function `MultiRayAirIceRefraction::FillInAi
rRefractiveIndex()':
```

⋮



```
MultiRayAirIceRefraction.cc:(.text+0xb0bf): undefined reference to `gsl_interp_accel_alloc'
/cvmfs/icecube.opensciencegrid.org/py3-v4/RHEL_7_x86_64/spack/opt/spack/linux-centos7-x86_64/gcc-4.8.
5/binutils-2.31.1-lh5ncjznzy34ztyisj5w2aytsqhz47i5/bin/ld: MultiRayAirIceRefraction.cc:(.text+0xb0d0)
: undefined reference to `gsl_interp_cspline'
/cvmfs/icecube.opensciencegrid.org/py3-v4/RHEL_7_x86_64/spack/opt/spack/linux-centos7-x86_64/gcc-4.8.
5/binutils-2.31.1-lh5ncjznzy34ztyisj5w2aytsqhz47i5/bin/ld: MultiRayAirIceRefraction.cc:(.text+0xb0e0)
: undefined reference to `gsl_spline_alloc'
/cvmfs/icecube.opensciencegrid.org/py3-v4/RHEL_7_x86_64/spack/opt/spack/linux-centos7-x86_64/gcc-4.8.
5/binutils-2.31.1-lh5ncjznzy34ztyisj5w2aytsqhz47i5/bin/ld: MultiRayAirIceRefraction.cc:(.text+0xb105)
: undefined reference to `gsl_spline_init'
/cvmfs/icecube.opensciencegrid.org/py3-v4/RHEL_7_x86_64/spack/opt/spack/linux-centos7-x86_64/gcc-4.8.
5/binutils-2.31.1-lh5ncjznzy34ztyisj5w2aytsqhz47i5/bin/ld: MultiRayAirIceRefraction.cc:(.text+0xb227)
: undefined reference to `gsl_spline_eval'
collect2: error: ld returned 1 exit status
make[1]: *** [corsika] Error 1
make[1]: Leaving directory `/ada_mnt/ada/user/sdekockere/corsika-77500/src'
make: *** [install-recursive] Error 1
```

Once this point has been reached, you can finish installation by following these steps:

- Change directory to the *src* directory.
- Copy the line that starts with */bin/sh ../libtool –tag=F77* and ends with *-lstdc++* from the *coconut* output (i.e. the first line of the figure above, but make sure to copy your line and not the one from the figure).
- Enter the line in your terminal, add *" -lgsl"* at the end, and execute the command.
- Copy the *corsika* executable in the *src* directory to the *run* directory.