A. Simulation of the incoming beam to GRACE.

The simulation program documented here is called **DegraderSimu** and it simulates the interaction of the antiprotons in the degrader foil, air gap, and vacuum windows between the AD and GRACE.

The **DegraderSimu** program uses the Geant4 framework [22], and for the work in this thesis version Geant4.10.4 was used. It is important to use this version or newer version, since the energy loss model in older versions had a bug in the energy loss model that only revealed itself when using the single scattering model at low energies [48]. The bug was found and resolved in collaboration with the Geant4 team when data was compared to simulations for the flux on the end plate as described in chapter 4.

The Geant4 framework is commonly used in high energy physics to simulate the interaction of particles with matter. By providing the geometry, material, physics list, and the definition of the particles of interest, Geant4 can simulate the passage of these particles through the geometry. Interactions between the particle and the material such as energy loss, scattering, and annihilations, are simulated using the Monte Carlo method. Information about the state of the particle can be obtained at any point of the geometry.

The **DegraderSimu** simulates antiprotons of energy 5.3 MeV coming from the AD, this energy corresponds to the energy of the AD-beam, and tracks them through the three foils and the air that separates the vacuum of AD and GRACE. This geometry is described in section 4.2. The antiprotons are scattered, loses energy and is possibly annihilated in the foils and the air gap. The output of the simulation is the energy, position and momentum direction right after the particles have entered GRACE through the circular window with a diameter of 4 cm. For the **GraceBeam** simulation these parameters has to be known, and therefore this simulation provides the input to the **GraceBeam** simulations documented in appendix B. The degrader thickness is an input parameter to the simulation and can therefore easily be altered.

A.1. How to run the simulation

This section shows step by step how **DegraderSimu** can be installed and ran. As a prerequisite Geant4 needs to be installed. Geant4 can be downloaded from `http://geant4.cern.ch` and installed according to the instructions found there.

A.1.1. Installing the simulation package

Before we start the Geant4 environment has to be set.

```
source <path>/geant4.sh
```

Then make a folder to run all the simulations in and make a build directory in this folder.

```
    mkdir DegraderSimulation
    cd DegraderSimulation
    mkdir build-GraceDegrader
```

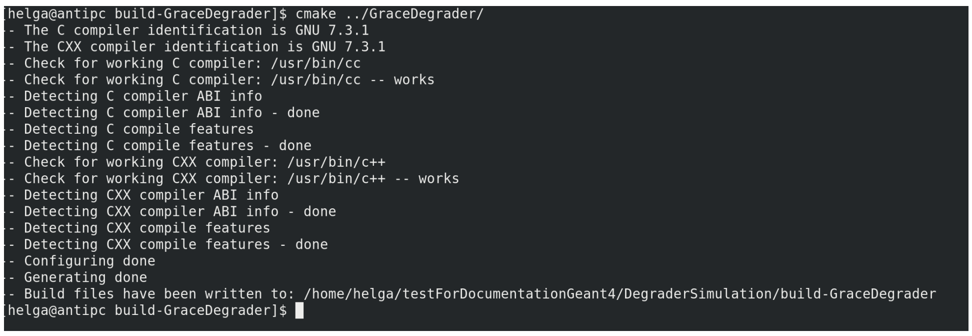
Clone your own version of the code from github, checkout the tag `v1.2` and see that the files and folders are there:

```
git clone https://github.com/helgaholmestad/GraceDegrader.git
git checkout v1.2
cd GraceDegrader
ls
AntiPcells.cc  CMakeLists.txt  include  instructions  makefile  src  vis.mac
```

The simulation can then be configured in the build-GraceDegrader folder using cmake:

```
    cd ../build-GraceDegrader
    cmake ../GraceDegrader
```

A successful cmake build should show an output similar to this:

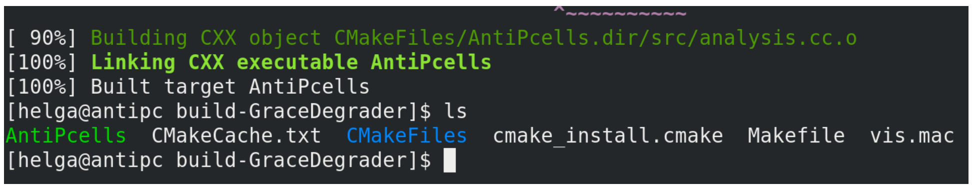


```
[helga@antipc build-GraceDegrader]$ cmake ../GraceDegrader/
- The C compiler identification is GNU 7.3.1
- The CXX compiler identification is GNU 7.3.1
- Check for working C compiler: /usr/bin/cc
- Check for working C compiler: /usr/bin/cc -- works
- Detecting C compiler ABI info
- Detecting C compiler ABI info - done
- Detecting C compile features
- Detecting C compile features - done
- Check for working CXX compiler: /usr/bin/c++
- Check for working CXX compiler: /usr/bin/c++ -- works
- Detecting CXX compiler ABI info
- Detecting CXX compiler ABI info - done
- Detecting CXX compile features
- Detecting CXX compile features - done
- Configuring done
- Generating done
- Build files have been written to: /home/helga/testForDocumentationGeant4/DegraderSimulation/build-GraceDegrader
[helga@antipc build-GraceDegrader]$
```

The simulation program can now be compiled:

```
     make
```

You will see some warnings about unused variables, but this causes no problems. If the compilation ends with the following lines the simulation should be properly installed.



```
                                                  ^~~~~~~~~~~
[ 90%] Building CXX object CMakeFiles/AntiPcells.dir/src/analysis.cc.o
[100%] Linking CXX executable AntiPcells
[100%] Built target AntiPcells
[helga@antipc build-GraceDegrader]$ ls
AntiPcells  CMakeCache.txt  CMakeFiles  cmake_install.cmake  Makefile  vis.mac
[helga@antipc build-GraceDegrader]$
```

A.1.2. Running the simulation

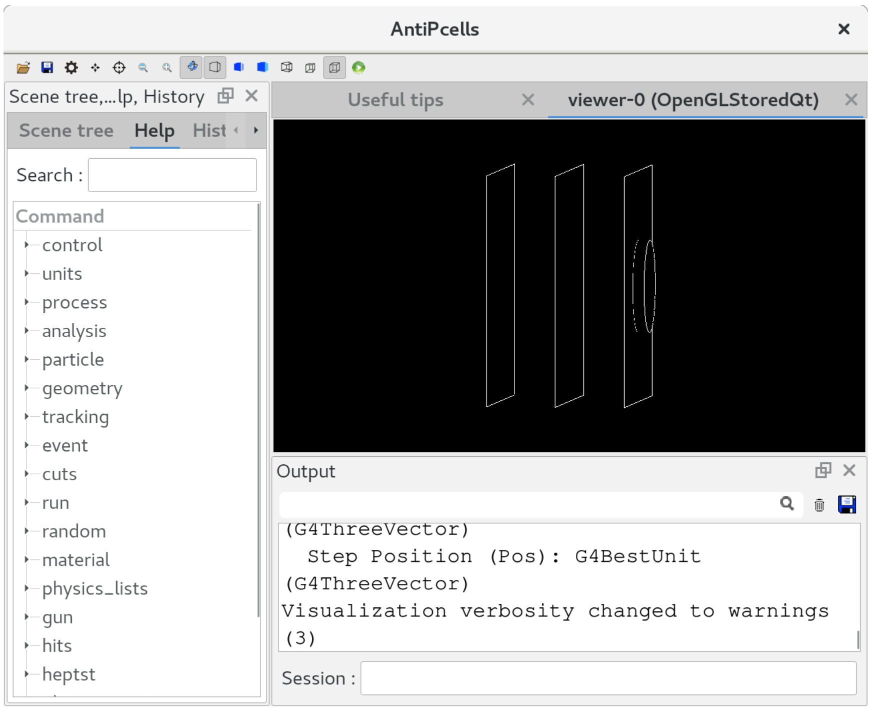
The program is run in the build-GraceDegrader folder, and can be run in interactive or batch mode. To run it in the interactive mode do:

```
./AntiPcells <degraderThickness>
```

The unit of the degrader thickness is µm, and for the experimental setup where the majority of the data was taken the degrader thickness was 33 µm. This corresponds to running the command:

```
./AntiPcells 33
```

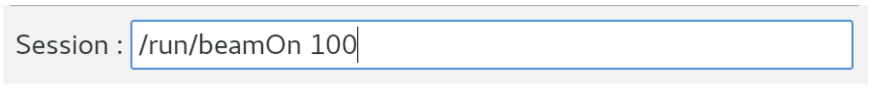
This mode will show a an interactive window with the geometry:



```
(G4ThreeVector)
  Step Position (Pos): G4BestUnit
(G4ThreeVector)
Visualization verbosity changed to warnings
(3)
```

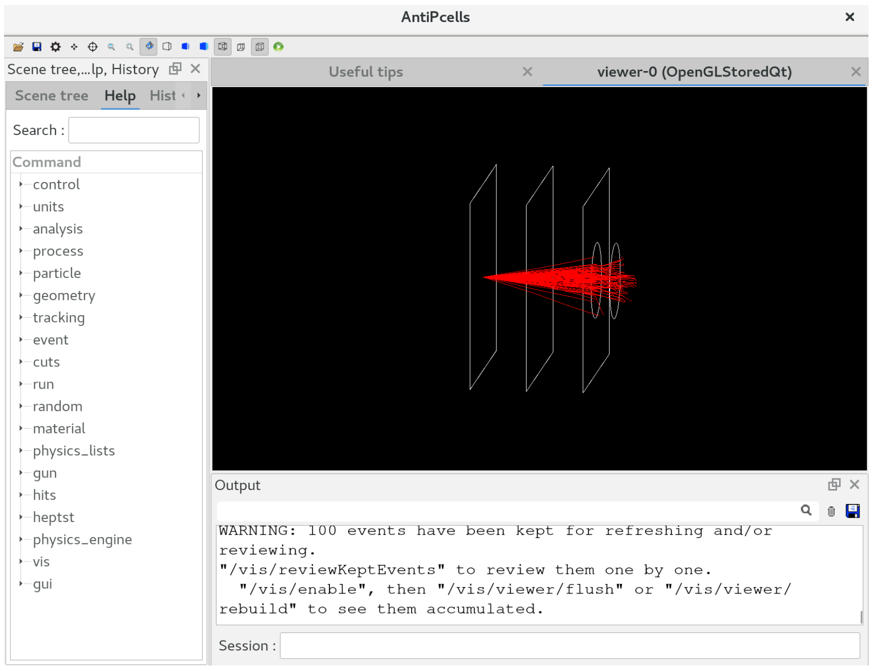
Here the three different foils are clearly seen, and the circular geometry corresponds to the window into grace and has a diameter of 4 cm. In the simulation the GRACE window is a sensitive detector, and we retrieve information about the antiprotons as they exit this volume.

The commands are written in the Session box. To simulate the tracking of 100 particles through the geometry do:



```
Session : /run/beamOn 100
```

Then the tracks of the antiprotons will appear:



```
WARNING: 100 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.
```

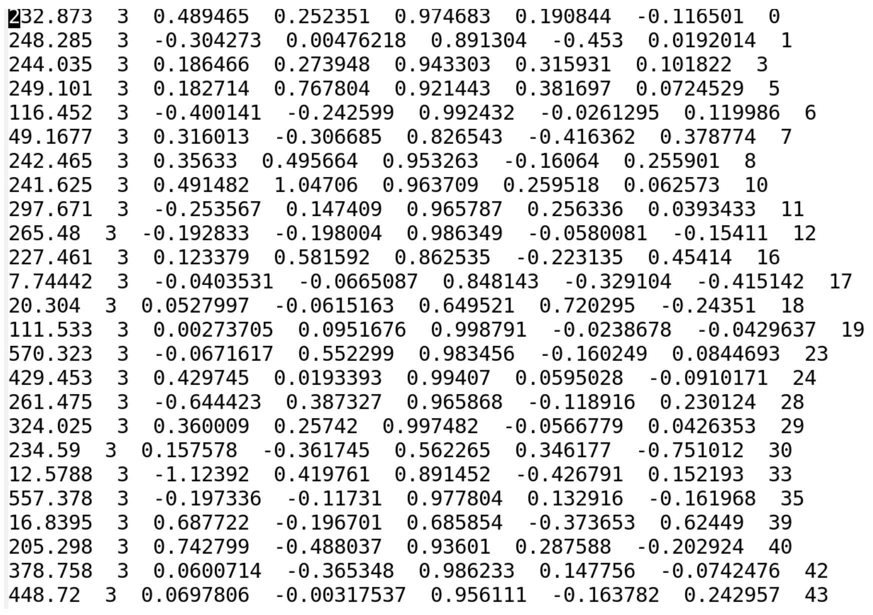
For simulations with many particles it is better to do it in the batch mode as the graphics is computationally heavy. To run in batch mode insert the commands that you would write in the SessionBox into a text file. Then the simulation can be run from the command line:

```
./AntiPcells <degrader thickness> <name of input file>
```

An example of an input file is included in the repository and the example can be run as:

```
./AntiPcells 33 ../GraceDegrader/input.mac
```

The output of the simulations in both modes is a file called texttDegrader.tex. This file has eight space separated columns and here is a sample of this file:



```
232.873  3  0.489465  0.252351  0.974683  0.190844  -0.116501  0
248.285  3  -0.304273  0.00476218  0.891304  -0.453  0.0192014  1
244.035  3  0.186466  0.273948  0.943303  0.315931  0.101822  3
249.101  3  0.182714  0.767804  0.921443  0.381697  0.0724529  5
116.452  3  -0.400141  -0.242599  0.992432  -0.0261295  0.119986  6
49.1677  3  0.316013  -0.306685  0.826543  -0.416362  0.378774  7
242.465  3  0.35633  0.495664  0.953263  -0.16064  0.255901  8
241.625  3  0.491482  1.04706  0.963709  0.259518  0.062573  10
297.671  3  -0.253567  0.147409  0.965787  0.256336  0.0393433  11
265.48  3  -0.192833  -0.198004  0.986349  -0.0580081  -0.15411  12
227.461  3  0.123379  0.581592  0.862535  -0.223135  0.45414  16
7.74442  3  -0.0403531  -0.0665087  0.848143  -0.329104  -0.415142  17
20.304  3  0.0527997  -0.0615163  0.649521  0.720295  -0.24351  18
111.533  3  0.00273705  0.0951676  0.998791  -0.0238678  -0.0429637  19
570.323  3  -0.0671617  0.552299  0.983456  -0.160249  0.0844693  23
429.453  3  0.429745  0.0193393  0.99407  0.0595028  -0.0910171  24
261.475  3  -0.644423  0.387327  0.965868  -0.118916  0.230124  28
324.025  3  0.360009  0.25742  0.997482  -0.0566779  0.0426353  29
234.59  3  0.157578  -0.361745  0.562265  0.346177  -0.751012  30
12.5788  3  -1.12392  0.419761  0.891452  -0.426791  0.152193  33
557.378  3  -0.197336  -0.11731  0.977804  0.132916  -0.161968  35
16.8395  3  0.687722  -0.196701  0.685854  -0.373653  0.62449  39
205.298  3  0.742799  -0.488037  0.93601  0.287588  -0.202924  40
378.758  3  0.0600714  -0.365348  0.986233  0.147756  -0.0742476  42
448.72  3  0.0697806  -0.00317537  0.956111  -0.163782  0.242957  43
```

The different columns is described in table A.1. The x position will always be at 3.0025 cm, since this is where the detector is placed. The momentum is given as the proportion of a unit vector, such that the length of the momentum vector is always 1.

Column number	Description
1	Energy of the particle [keV]
2	x-position [cm]
3	y-position [cm]
4	z-position [cm]
5	x-momentum direction
6	y-momentum direction
7	z-momentum direction
8	Event number

Table A.1.: Description of the different columns in the output file texttttDegrader.tex.

A.2. Description of the main features of the simulation

This section is meant to describe what the different classes do in the program. The overview is meant to help the reader to understand the the work flow of the simulation and be able to customize it if necessary.

A Geant4 program is a framework written in C++. It therefore contains a main function steering the program, which is found in the file `AntiPcells.cc`. From this function different instances of classes are called at different points of the simulation. Table A.2 contains all user defined classes that are called during the simulation, together with a brief description of what they do and when they are called. All special considerations that had to be taken into account because we were working with a low energies, thin foils, and antiprotons are mentioned.

DetectorConstruction	Called at the start of the program. Defines the geometry. Usually also defines on or more sensitive detector. A sensitive detector is the area of the geometry where the user is interested in retrieving information from. In this program the sensitive detector is a circular area with diameter 4 cm right after the GRACE foil.

PhysicsList	An instance of this class is called as the program starts to define the interaction processes between the material and the particles. Since we are using a pre-made physics list this is done directly in the main function by:
	`G4PhysListFactory factory;`
	`G4VModularPhysicsList* phys = factory.GetReferencePhysList("FTFP_BERT");`
	`phys->ReplacePhysics(new G4EmStandardPhysicsSS());`
	`phys->RegisterPhysics(new G4EmUserPhysics());`
	`runManager->SetUserInitialization(phys);`
	The list FTFP-BERT is used as this contains annihilation of antiprotons. We are also forcing the use of single scattering instead of multiple scattering, because multiple scattering is not accurate enough for the very thin foils.

G4EmUserPhysics	As default the antiprotons annihilate when they reach an energy below 1 keV. This user implementation sets this threshold down to 10^{-6} eV.
RunAction	A run is defined as the set of particles that are simulated, and functions in this class are called at the start and end of the run. The file `Degrader.tex` is created at the start of the run and closed at the end.
PrimaryGeneratorAction	Defines the incident particles. In our setup this is a simple 5.3 MeV point beam at position (-3 cm,0,0) with momentum vector (1,0,0).
EventAction	An event is defined as the simulation of one particle trough the geometry. An event always starts with generating a particle as defined in the PrimaryGeneratorAction class. It is implemented that at the end of each event information about the antiprotons as they go out of the Grace window is written to file.
SteppingAction	The particles iterate through the material in steps, and at each step a stochastic process such as energy loss or multiple scattering can happen. In the program it is implemented that an event is aborted when the antiproton annihilates. This saves large amount of computation time since we avoid tracking all the annihilation products.
AntiPSD	An instance of this class is called every time a particles makes a step in the sensitive detector. Information about the energy, momentum and position of the particle is here added to the hit collection.

AntiPHit	The AntiPSD class calls functions in this class to add information to a hit collection. There is one hit collection for every event.
analysis	Singelton class that takes care of writing the interesting information from the hit collection to file. Here the actual implementation of writing to file is found, and functions in this class is called from the classes runAction and EventAction.

Table A.2.: Description of the different classes in the simulation program. Information about when functions in the classes are called, what they do and how they are customized to low energy antiprotons in thin foils.