1 Geant4

Geant4 is a toolkit used for the Monte Carlo simulation of particle interaction with matter. Monte Carlo is a general numerical integration technique that has constant convergence irrespective of the number of dimensions a problem is in. Generally speaking, particle transport is a very high dimensional problem, and therefore Monte Carlo simulation is the best way to solve it.

Particle transport is a problem which has been approached a number of ways in other codes, but the method used in Geant4 is similar to that used in an older code called EGS4. Particles are transported in discrete steps, the length of which is determined by the processes applicable to the particle. Boundaries in the geometry are handled by a process as well, which may interrupt the tracking to traverse a boundary correctly.

Which processes are applicable to a particle is completely up to you, but I have written this application to use the 'standard' medical physics processes, which have been tested by others to be acceptable. Another factor to choose is the 'cut-value' which is essentially a threshold for secondary particle production, and can have considerable effects on the accuracy of the simulation. Again, I have set these values to the accepted best practise ones, but you may like to play with them. All of these settings can be controlled from the user interface, or by directly editing the "Physics.cc" file and re-compiling.

There are a lot of subtleties in the physics within Geant4, and I recommend you read the relevant sections in the Physics Manual.

Geant4 can simulate pretty much any particle you want, but you'll be mainly interested in protons and electrons. I have written the program to take advantage of a piece of code within Geant4 called the General Particle Source. This allows you to look at the energy deposition of any type pf particle, at any energy you desire. You can also specify distributions to sample energies from, for example it is common to assume a Gaussian energy spread in beam energy, which has a considerable effect on the dose distribution. it is also possible to define the shape of your source, allowing you to model semi-realistic beam spots is you so wish. All of this can be done from the interactive command prompt, or through the use of macro files.

Provided you install it correctly, Geant4 can also provide real-time visualisation of the particle tracks in your simulation. This is very useful for making sure that your geometry is right, but will significantly slow down the simulation if you are running for results, so don't forget to turn it off.

Geant4 is tricky, and this is by no means an exhaustive description. For more information, there are the Geant4 Physics Manuals, and the Geant4 Application Developer's manuals, both of which will explain everything in great detail.

2 The Geant4-Generic program

This program simulates any kind of incident radiation on a block of water and saves the energy deposition profile in a histogram. The histogram may be 1D,

2D or 3D. In the case of 1 and 2D, the energy deposition profile is flattened, and does not represent a slice through a 3D histogram. If you want that, you'll have to look at post-processing of a 3D histogram in python or something.

The histograms are controlled from the UI interface, or macro files. I'm afraid that only one can be used at a time at the moment, but if you really really need to be able to do more than one, I can have a look at doing it.

Histograms are saved in binary files which can be read into other programs. My favourite language for data processing is python, but you can use whatever you want. 1D histograms are written in order from bin 0 to bin N, so each double in the output corresponds to the energy deposited in that bin. In 2D, bins in x are contiguous, while bins in y are offset by N_x. This is difficult to explain in words, but I can draw a picture. In 3D, a similar offsetting is used, but in 3 dimensions - this is as specified in the Analyze 7.5 format for medical images, something you should probably look up.

The histograms are written using the C++ std::map class, which compresses the data used at runtime at the expense of performance. I did this because I figured you would probably be running the simulations on your own computers which might not necessarily have huge amounts of memory. If you do happen to have access to something with about 40GB RAM (depending how many threads are used...) then I can switch the LowMemHist class out for the 'normal' one, which is a bit faster but uses much more memory.

When you build Geant4 make sure to turn Multithreading ON, this will allow you to make full use of the bigger computers we have in the department by running with, say, 48 threads (I wouldn't make a habit of that though - people will get angry with you). The frontend code is written in such a way that it will work with single threaded mode, but seeing as the work is already done, you might as well have the speed boost from multithreading.

I have written a class to interface most things you will need to do with the Geant4 UI commandline, which can be accessed by running in interactive mode. This is quite cool, and can give you semi-interactive help on any command within Geant4. To invoke it, run an interactive session and type "help" at the prompt. Navigation is by numbers, but make sure you don't send an EOF signal (CTRL-D) to the interpreter, otherwise it will crash spectacularly. This is something I'm trying to fix...

To get the code for the frontend, you will need to use git, which is a version control system. Using this, you can execute a 'clone' command to pull down a copy of the code from my github repository. If you find bugs or really need extra features, I can sync the repository after I've made changes, and then you can pull down the changes and rebuild. It should work smoothly. I'm going to assume you have git installed, but if you don't it should be easy enough to get hold of. The command you will want to execute is:

```
git clone https://github.com/afg1/Geant4-Generic.git
```

This will download all the code to a folder named Geant4-Generic on your computer. If you then enter the directory and create a subfolder called 'build'

or something like that, you can build the code without screwing up the source tree.

Building the executable requires CMake (preferably CCMake - the commandline GUI), a C++ compiler and GNU Make. This is of course assuming you're running under Linux/OSX. If you're running on Windows, I suggest you find a computer with Linux on it, because Geant4 is less well supported on Windows, and I have no experience whatsoever running under Windows and can't help.

I recommend GCC-4.8, which you should have on a reasonably recent Linux install, and shouldn't be too difficult to get hold of on Mac. There are instructions for Windows, but as I said, I have no experience in the matter, so you're on your own if that's the way you want to go.

To build the code, follow these steps:

```
cd <build directory>
ccmake <path to source directory>
```

This will bring up the CCMake GUI, the only thing you will need to change is the build type. To do this highlight it and hit enter, then type 'Release' (this sets flags for maximum optimization) and hit enter again, then hit 'c' to configure, wait for it to finish, then 'g' to generate. You will then exit the CCMake GUI and return to the commandline, where you can execute

```
make -jN
```

where N is the number of threads you want to use. There is no point having N > 10, because there aren't that many files to compile. The result of this will be an executable file called G4e in the build directory, this is what you want.

For further details, see the README file in the repository, which also has a full description of the messenger classes