Geant4 14keV Phosphorus Ion Silicon/Platinum Aperture Simulation

Jonathan Newnham

October 17, 2010

Contents

1 Introduction 2

2 Device to be Simulated 2

3 Development Notes 2
- 3.1 Ion Scattering . . . 2
- 3.2 Next steps . . . 3
- 3.3 Development . . . 4
  - Extracting Tracks . . . 4
  - Geometry . . . 4
  - General Particle Source . . . 4
- 3.4 Comparison with SRIM . . . 4
- 3.5 Edge reflection . . . 5

4 Geant4 Usage 6
- 4.1 Installation . . . 6
- 4.2 What to do with it? . . . 6
  - Python . . . 6
- 4.3 Documentation . . . 7

5 Program 8
- 5.1 Compilation . . . 8
- 5.2 Debugging . . . 8
- 5.3 Commands . . . 8
- 5.4 Explanation . . . 8
- 5.5 Visualisation . . . 9
- 5.6 Output . . . 9
- 5.7 Geometry Definition . . . 9
- 5.8 Further Development . . . 10

6 Simulations Conducted 10
- 6.1 SRIM comparison . . . 10
- 6.2 Trapezoidal aperture . . . 10
- 6.3 More detailed apertures in Silicon and Platinum . . . 10

6.4 10keV Phosphorus simulation 11
6.5 Varied-parameter simulations 11
6.6 Diamond Masked Implant simulations 12
6.7 Aluminium Oxide Nanotemplate simulations 12
6.8 Silicon nanowire implantation density 12

1 Introduction

I have been tasked with simulating Andrew's and Jess' method of implanting 14keV phosphorus through an aperture. I will use the Geant4 toolkit[1]. The previous simulation, based on TRIM, allowed simple aperture geometry (cylinder or rectangular) but required extensive programming to change the aperture geometry. It did not give very good results.

The simulation is also later adapted for different devices, such as a masked diamond implant and an aluminium oxide template with many holes in it.

2 Device to be Simulated

The physical device to be simulated is constructed in the following way. An approximately linear hole is etched in a 2μm silicon cantilever. Its dimensions are 2μm long and deep, and it is 400nm wide at the top and 200nm wide at the bottom. The interior of the hole is then coated in platinum until the gap is about 50nm wide[2]. 7keV to 14keV singly-charged phosphorus ions are then fired at the aperture, at up to 3° from normal incidence. 0.5MeV helium nuclei are also used.

We want to know if many scattered (slowed) ions travel through the aperture successfully. Scattered ions may end up in the wrong place, and slowed ions will stop at a different depth. Significantly slowed ions will not be noticed on the single-ion-detection equipment.

The aim is to implant hundreds or thousands of ions deterministically with this technique.

3 Development Notes

3.1 Ion Scattering

The ion-scattering part of Geant4 was developed to provide the same capabilites as SRIM in this area but with all the complicated geometry[3] and other processes power of Geant4 also available. The processes that light-ion scattering will need are taken from the Geant4 Physics Reference Manual (version 9.2). This part seems quite focused on ions with Z $<$ 3.

The article [Mendenhall and Weller, 2005] says Ziegler-Biersack-Littmark screening is reasonable for many nuclear straggling and implantation problems. The article provides an experimental comparison to SRIM demonstrating reasonable agreement[4]. Nuclear and electronic scattering are both important processes that we want to model. That article makes references

[1] the architecture, documentation and attention to detail with this toolkit are truly impressive.
[2] at unknown depth - this data comes from a top-down view. Also only one part along the length is this wide - one end might be closed and the other open.
[3] SRIM can only handle planar layers of material - the material can only depend on the X coordinate.
[4] SRIM is not perfect but widely used.

to the ScreenedCompton physics process in Geant4, and the Nuclear Stopping Powers component of the `G4hLowEnergyIonisation`[5] class sounds something like what we need, although it looks like it hasn't been changed since 2002.

The algorithm is not implemented by the Geant4 base classes.

The Geant4 Twiki says[6] that as of version 4.9.3.b01 the `G4hLowEnergyIonisation` class for hadron ionization is not maintained anymore and should not be used. Instead, use the `G4ionIonisation` class provided by the Standard EM working group. This is invoked by

```
processManager->AddProcess(new G4ionIonisation(),-1, 2, 2);
```

This process includes nuclear scattering by default[7] for protons below energy 2MeV. Ions heavier than protons have their energy cutoff scaled accordingly. However it only applies to charged particles.[8] It also looks important to define materials as (one of the 78) NIST materials for which low-energy stopping power data is available. Silicon is one of these; platinum is not. The only other process which will be needed is the transport process, as far as I can tell.

There is a note that says that for 1keV protons, this method is only accurate to about 20%. Presumably it gets considerably worse at lower energies.

Extended test `/extended/electromagnetic/TestEm7` deals with heavy ions. Use it to learn syntax for generating ions with a particle gun. It also includes Mendenhall's code, a cleaned up version of what he used to write [Mendenhall and Weller, 2005].

Examples `underground_physics` and `microdosimetry` make use of the GetIon method.

Jess has alerted me to the article [Paul and Schinner, 2003] which compares the accuracy of several ion simulation toolkits.

(2010-01-27) - Well, Geant 9.3 has been released. Mendenhall's G4ScreenedNuclearRecoil process is included. It doesn't model electronic stopping power.

The G4hLowEnergyIonisation class models nuclear stopping using a universal parameterisation (which can be disabled and doesn't produce recoils); I think it should be ok to use this to model electron stopping power and use G4ScreenedNuclearRecoil to model nuclear stopping power. Actually scratch that. G4hLowEnergyIonisation has been deprecated; the StandardNR physics list includes the replacement, G4ionIonisation. Phew. No wonder it worked.

3.2 Next steps

The next thing to do is to run a simple simulation (particlegun, block silicon target) and compare the range and scattering with an SRIM simulation.

To do this I will need to recover the final position and momentum of particles. If they have momentum above a cutoff (say 10eV) they are probably transmitted ones, otherwise they are stationary. I will plot histograms of depth and horizontal scattered range and compare them to SRIM.

Plotting energy lost to ionisation is harder – there is more data to deal with (a long train of data for each ion). This data can be retrieved using a Get method of the G4ParticleChange class.[9] I will not do this for now.

[5] see G4 Physics Reference Manual, §12.10.9

[6] https://twiki.cern.ch/twiki/bin/view/Geant4/LoweMigratedProcesses

[7] Physics Reference Manual §9.1.2, Nuclear Stopping Power section.

[8] I'm not sure if this is a problem. Prof. Jamieson indicated that ions quickly pick up an extra electron and become neutral while travelling through a material. I'm not sure if this matters – I should find out what, if anything, SRIM does about this. Actually, when you specify particles for the particlegun to fire, you specify the excitation energy. TestEm7 uses 0 for this; I assume this means the "ion" is not charged. I do this too. Should probably check all this at some point.

[9] G4 Book for Application Developers §5.1.2

If the simulation is not accurate, some things to try (in order of increasing difficulty) are:

1. change range values - control cutoff energy for secondary particle generation (?)
2. investigate scattering events - collisions should produce secondary particles, ensure this is the case
3. Try and incorporate Mendenhall's code from TestEm7 (see above).

3.3 Development

Actually, TestEm7 basically does everything necessary, and as a bonus it has Mendenhall's screened nuclear stopping built in. I think I will just modify this example to do what I want. I have succeeded in changing it to 10keV P-31 ions by modifying the proton macro. I just have to change the material to NIST silicon (`G4_Si`). A list of NIST materials can be obtained by the command[10] `/material/nist/listMaterials all`.

Now I must work out how to change the geometry of TestEm7 (shouldn't be too hard..) and extract tracks (terminating them at the boundary?).

I use KDevelop to edit the code. A few minutes effort was not enough to import the example as a proper project.

Extracting Tracks

G4UserTrackingAction looks like a good place to extract final position/energy information. It was. A small amount of code writes the final positions of each particle to a finalPositions.txt file in the current directory. Values are x (nm),y (nm),z (nm),finalEnergy (eV). Transmitted/backscattered particles will have nonzero energies, stopped particles will not.

At this point I compared output to SRIM. See Figure 3.1 and §3.4.

Geometry

Upon reflection, I think the easiest way to model a slit is with CSG. I will use the same slab of silicon as before but subtract away the slit shape.

This method makes it easy to model the slit dimensions and also easily allows a finite-length slit. It also permits for easy rotation of the slit.

On top of all that, the maths to work out coordinate points is much simpler than trying to project up two sides of the slit.

General Particle Source

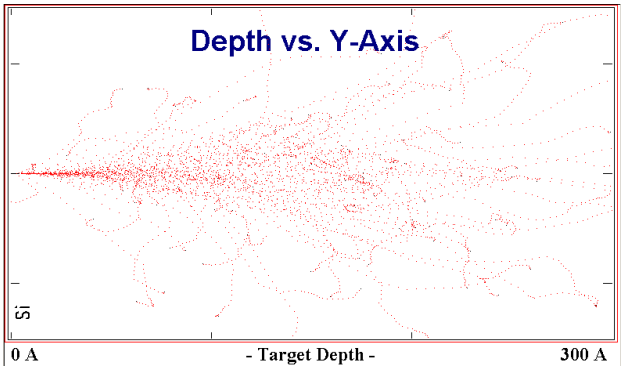
Turns out this thing is quite hard to use. Never Mind!

3.4 Comparison with SRIM

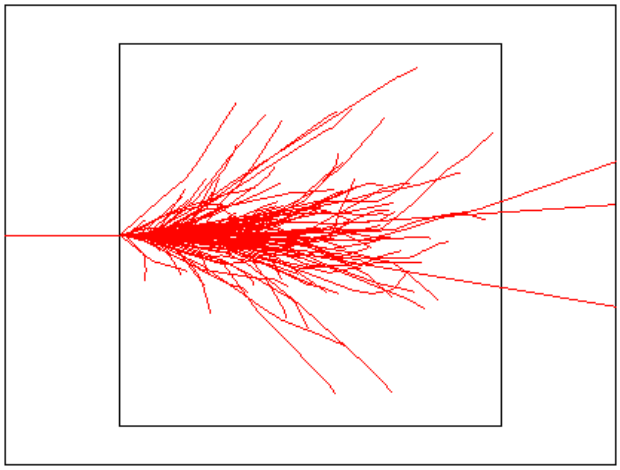
SRIM is a widely-used method of modelling ion implantation. It is very easy to use. This section compares the Geant4-StandardNR model of TestEm7 to SRIM's "full damage calculation" model.

[10] run at the Geant4 command line (the "G4UI"). Get to this by running an example with no parameters. TestEm7 is a good one!

This section was done with range cuts and maximum step sizes of 0.1 nm. The standard value is 1 mm. Using a value of 1mm did not make a statistically significant difference.[11] Such a larger step size is better because the simulation runs in about half the time.



(a) SRIM particle tracks



(b) Geant4 particle tracks (with StandardNR physics list). The inner box is the Si target; it is 30nm wide.

Figure 3.1: Comparison of Geant4 and SRIM. The tracks look reasonably similar.

Geant4-SNR	Range	Straggle	SRIM	Range	Straggle
Longitudinal	14.0	7.3	Longitudinal	16.0	6.9
Radial	4.4	3.4	Radial	8.7	5.0
Transmitted	5.5%			8.9%	

Table 1: Comparison of Geant4 and SRIM for 1000 10keV P-31 ions into Si. SNR indicates that the StandardNR Physics List of TestEm7 was used. Values are in nm.

3.5 Edge reflection

A 14keV phosphorus ion has a de Broglie wavelength of 0.9Å. Crystalline silicon has a lattice spacing of 0.3Å. Hence at low angles, the ion will "see" many silicon atoms before it penetrates one crystal plane's width into the crystal. This means that reflection from the inner surface of the aperture needs to be considered. The reflection will be very noticeable because the scattering angle is so small[12]. I am assuming that the Monte Carlo approach used will replicate this effect accurately. (Note: It does. Thin-film reflection has been modelled successfully with GEANT4).

[11] For 1mm step sizes, ('Longitudinal range is', -0.29378445102019213) ('Longitudinal straggle is', 7.2128396681415063) ('Radial range is', 4.6094302586809759) ('Radial straggle is', 3.3771854025951895) (59, 'transmitted ions.')

[12] a maximum of 3° for a single reflection path as the aperture is about 100 times as deep as it is wide

4 Geant4 Usage

4.1 Installation

I tried installing under Ubuntu 9.04 on the unimelb baker machines.

Dependencies:

- clhep – you'll probably have to download and compile this yourself
- build-essential
- xerces (an XML parsing library – only for GDML support, the c variant)
- libboost (C++ library – only for g4py, the python interface)

I used the Geant4.9.3.b01 version. It only runs on Linux. I got an example running on the baker machines after several hours work. Installation instructions can be found on-line.[13] While running the `./Configure -build` script, enable the shared libraries (also build the normal kind) but default to granular, and avoid the Qt stuff, there are compile errors under Ubuntu (or there were in 4.9.3beta, that's supposed to be fixed). Choose to collect all header files into one folder, and execute `make includes` in the `source` directory after compilation has finished. Make sure everything's okay by running an example. Novice N01 is a quick one.

Setting environment variables properly is very important for compilation and execution. A script env.sh (generated by running Configure without the -build option) does most of this for you. I've put other settings into `setenv.sh` - setting LD_LIBRARY_PATH to include the CLHEP and xerces libraries, and setting the PYTHONPATH to include the `g4py/lib` directory.

The other thing I had trouble with (I had to compile xerces myself as I don't have apt-get access on the baker machines) was xerces linking to something called libicu. There were seven libs that I needed to copy into geant4.9.3/lib/Linux-g++ : libicu{data,i18n,io,le,lx,tu,uc}.so.40 . I can't remember where I found them - good luck! They're probably in an Ubuntu package somewhere.

4.2 What to do with it?

There are lots of different ways of using Geant4. C++ programming, python programming or using MODO.

The first way is writing a C++ program based on the Geant4 framework, and using it from the command line.

MODO is a java GUI that generates C++ files specifying geometry and physics processes, taking much of the work out of the C++ programming route. Some programming is probably still required.

Python

The other way is to use the Python bindings in the `environment/g4py` folder. This requires compiling Geant4 as a shared library.[14] To do this, delete the /tmp folder and run `./Configure` again, choosing to build global shared libraries (`.so` files). Or if you followed instructions above,

[13] http://geant4.slac.stanford.edu/tutorial/installation/Geant4.8.3/Linux/Geant4_8_3_Linux_Installation.htm#_Doing_the_Initial

[14] http://geant4.cern.ch/UserDocumentation/UsersGuides/ForApplicationDeveloper/html/apas08.html

you've already done it this way. Then set up your paths properly and `./configure` in the `$G4INSTALL/environments/g4py/` directory.

The g4py configure script is broken near the xerces stuff if you're trying to include GDML. Set the clhep and xerces paths manually with command-line parameters.

I have succeeded in compiling the Python API. Running "import Geant4" is quite cool! The API is not as well documented as the rest of Geant4.

4.3 Documentation

There is documentation for different aspects of Geant4 all over the web. Search for it.

The documentation I used:

1. Official documentation[15]. These can be found online through a search engine.
   a) Book for Application Developers – if you're taking the C++ route, this will be heavily used. It tells you how to write applications based on Geant4.
   b) Introduction to Geant4 – read this, I did
   c) GDML manual (has good diagrams of geometry construction). It's probably good practise to use GDML to describe your geometry - that way, it will be easier to transition to newer versions of Geant
   d) Geant4 Physics Reference Manual – For our purposes, we really only want the "low-energy" EM processes.
   e) Geant4 Software Reference Manual – Online only. Full specification of Geant4's classes, use if you don't know the arguments to a method or what methods a class makes available.
   f) Geant4 Book for Toolkit Developers – for really serious users, I haven't looked at this
2. The CERN Twiki is reliably up-to-date. In particular, the EM page[16] is useful, providing among other things information about "low energy" physics lists[17].
3. The Geant4 Python API documentation. Tells you how to compile the python API.
4. Someone's Geant4 Kubuntu Installation instructions[18] have some useful comments.
5. An article to prove it's possible[Gorelick et al., 2009].
6. An article about more accurate nuclear stopping powers. This is a good suggestion that hasn't been implemented in the Geant4 core as far as I can tell. [Mendenhall and Weller, 2005] Nuclear stopping is important for low-energy (<100keV) heavy (Z>3) ions.

One of the things that I had a lot of trouble with was the multitude of geometry viewers available. After a bit of experimentation I decided that HepRep was the best. If your program enables the Geant4 command line, heprep files can be generated by the commands

```
/vis/open HepRepFile
/vis/drawVolume
/vis/viewer/flush
```

A HepRep tutorial is available.[19] HepRep comes as a .jar file (Java: cross-platform, no installation necessary) and can generate vector images (.eps) for publication.

[15] http://geant4.cern.ch/support/userdocuments.shtml
[16] https://twiki.cern.ch/twiki/bin/view/Geant4/ElectromagneticPhysics#PhysicsLists
[17] A "physics list" is a predefined set of physics processes that saves you having to define one yourself.
[18] http://www.ece.ualberta.ca/~kirkwood/GEANT4-Kubuntu.html
[19] http://geant4.slac.stanford.edu/Presentations/vis/G4HepRAppTutorial/G4HepRAppTutorial.html

5 Program

The program consists of several C++ files that require the Geant4 source to compile. The code is in /home/jnnewn/locateIonVacWork/geant-baker/g4work/TestEm7 .

5.1 Compilation

Adjust setenv.sh if necessary. In a bash shell, cd to the code directory (g4work). Run the command

```
source setenv.sh
```

to set all your source environment variables. Now run

```
make
```

to compile.

The executable appears! (hopefully.)

5.2 Debugging

I used GDB for debugging. Set the environment variable G4DEBUG in the GNUMakefile to enable debugging symbols. Run the program to load most of the shared libraries, and set breakpoints using filenames and line numbers.

5.3 Commands

On a baker machine,

```
source /home/jnnewn/geant/setenv.sh # sets environment variables
cd ~/test # should contain p31.mac
TestEm7 p31.mac a
```

5.4 Explanation

Once you have the Geant4 application up and running, the Geant4 command line (the prompt is `Idle>`) is quite good. Type "`help`" to learn the commands. Commands can be placed in a batch file and executed with `/control/execute`. I have provided an example batch file `p31.mac`, found in /home/jnnewn/geant/g4work/TestEm7 . Additional detector parameters can be changed using commands in the `/testem/det/` command directory of the Geant4 terminal (discover them via "`help`"). Once you're partway through typing the name of a command, use tab to complete the command or Ctrl-D to list possible completions (suggestions).

You can run a macro when starting the application by providing the macro filename as an argument, i.e.

```
> TestEm7 p31.mac
```

By default, the program exits after the macro has finished. If this behaviour is not desired, use

```
> TestEm7 p31.mac a
```

(any second argument will present the G4UI command line when all the commands in the macro have finished executing).

5.5 Visualisation

HepRep is a reasonably intuitive and flexible visualisation option. It writes files which can be viewed with the HepRepr Java viewer. This viewer also exports vector images (good for publications). Vector images can also be exported by the OGLIX viewer's PrintEPS command.

The OGLIX viewer is good for quick feedback. It is the default in the provided macro file.

The `vis_rotate.mac` macro will run `p31.mac` and then orbit the result. Nicer movies can be made, google.

5.6 Output

The program appends to a file "finalPositions.txt" in the current directory. This file has one line for each fired particle, stating the final position and energy:

x (nm)	y (nm)	z (nm)	energy (eV)

Transmitted particles will have an x value that is the same as half the world volume width.

There may also be three extra columns giving the final direction (as a unit vector) of the particle. This is only useful if the energy is non-zero, and usually happens when a particle hits the bounding box of the simulation (the edge of the "World Volume".

5.7 Geometry Definition

The GDML manual has a good description, with some pictures, of how the conceptual basis for Geant4's geometry definition. For very complicated shapes Tesselated volumes are probably best; for the (initially) simple ones here, I used extruded faces. The basic geometry looks like Figure 5.1.

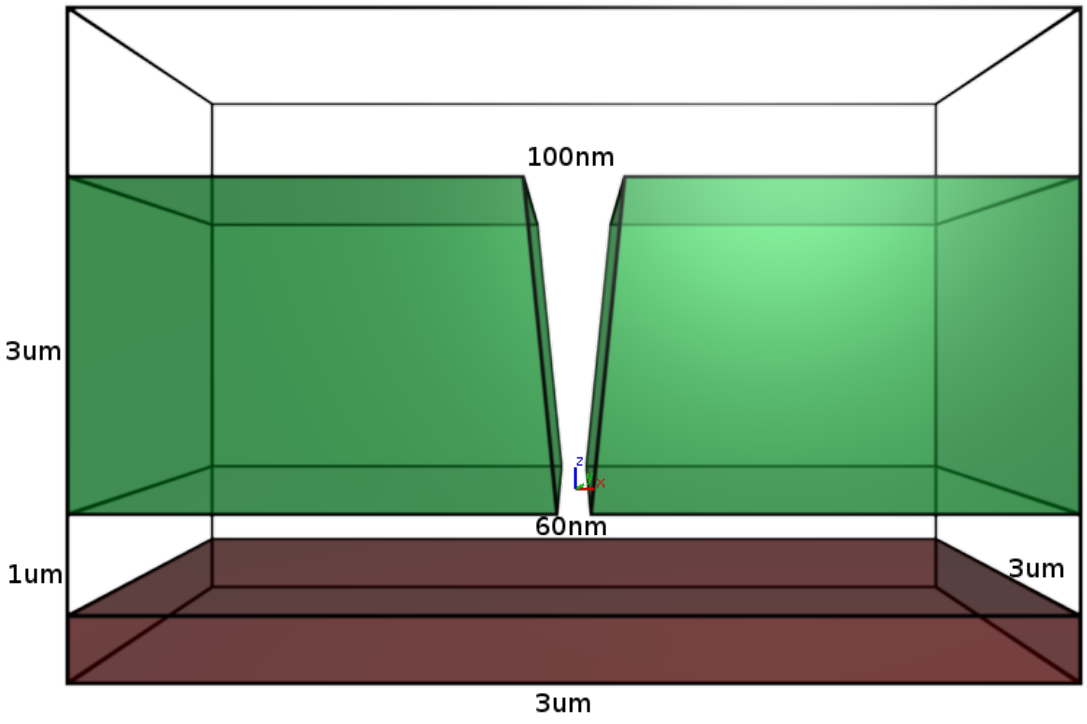


Figure 5.1: Basic Geometry of simulation apparatus. The green aperture is crystalline silicon; the red region at the bottom is the detection region.

5.8 Further Development

I originally developed this in the Geant4.9.3 beta, and migrated to version 9.3 when it was ready.

I started with the TestEm7 example and modified it. I maintained a history of the modifications in a Mercurial (.hg) repository.

I really only had to make significant changes to the DetectorConstruction class. This is where the geometry is defined; I used CSG (constructive solid geometry) because it visualises better, although sometimes the boolean operations stuff up the visualisation (very annoying - work around by perturbing geometry slightly).

Custom UI (Geant4 command line) commands are defined in the DetectorMessenger class, which gives instructions to the DetectorConstruction.

After some experimentation I found that an easy way to edit the source files with code completion was to open the directory in KDevelop and then find the errors when it couldn't find included files. Solve that error through the dialog's option to add custom include paths to the project and Geant4 include directories.

6 Simulations Conducted

6.1 SRIM comparison

The first step was to compare to SRIM by firing into a solid block of material. See Table 1. The results were quite satisfying at 10keV except that the Geant4 StandardNR model generates almost no secondaries whereas SRIM generates one for most collisions.

A quick look into this revealed that the ScreenedNuclearRecoil physics process should generate secondaries but that this can be disabled. Possibly this is something to do with the TestEm7 example; using the standard physics list might fix this. There is a separate implementation of ScreenedNuclearRecoil for the TestEm7 example (as that is where it originated).

6.2 Trapezoidal aperture

I then simulated a trapezoidal aperture in a silicon block and simulated firing 500keV He through it. This resulted in flat ratio of full-energy-transmitted vs. small-energy-transmitted ions when the apertures were aligned, and a linear drop to zero once the front aperture occluded the back one until there was no visible path from the point of view of the gun through the aperture.

6.3 More detailed apertures in Silicon and Platinum

I then tried to match the measured data more closely by constructing a more complicated aperture.

Jess makes the real apertures by milling a silicon cantilever. The top is 1 micron wide and the bottom 100nm. She then closes up the aperture using platinum. I changed the material of the more detailed aperture above to platinum. Platinum has about three times the stopping power of silicon (according to SRIM) and so is a good choice for making a barrier from! However the penetration of 10keV P-31 into either silicon or platinum is so small (for a 3 micron deep aperture, P-31 only penetrates $<$10nm) that this may not matter when we switch to low-energy heavy ions.

See Simulations in `/home/jnnewn/geant/results` on `tauon`.

Simulation 1 used an aperture shape (narrower after the the chokepoint) in Pt which resulted in an almost Gaussian angle-vs-full-energy-ratio plot (not linear!) and a very large shelf.

Simulation 2 used a silicon aperture, and produced lots of low-energy stuff but no shelf just below full energy; Simulation 3 was the same as 2 but in Pt, resulting in more of a shelf.

Simulation 4 narrowed the choke from 60 to 50nm to try and make the shelf taller relative to the undisturbed channel. Also inadvertently switched back to Si. Simulation 5 was the same, but in Pt.

Simulation 6 narrowed the choke to 30nm. Simulation 7 narrowed it to 5nm. Simulation 8 narrowed it to 30nm and widened the back end to 300nm.

I now consider that using silicon and platinum together in the same model may be important, as well as specifying different widths for each end of the aperture.

Simulations 9-11 tried to demonstrate the effect of a triangular obstruction of varying thickness and secondary-aperture size.

Simulations 12-17 refined the shape of the aperture to be more bulbous like Jess described. Simulation 17 matched the experimental results closely.

6.4 10keV Phosphorus simulation

All the above simulations were designed to increasingly approximate the experimental measurements on the aperture using 500keV He. These ions/energy are used because it is possible to detect where the particle hits the detector. The point of constructing the apertures is to guide 10keV P-31 ions. The experimental detector is not capable of recording the impact position of these ions as they have too low energy.

I repeated the platinum-lined aperture simulation using 10keV phosphorus, which is the final configuration for the experiment.

Simulation 18 used the refined aperture to simulate 14keV P-31 ions and found some scattering.

Simulation 19 flattened the important region of the aperture and found 0.07% scattering of the several thousand ions that made it through the aperture.

Simulation 20 unflattened the important region and found 0.3% scattering.

Simulation 21 went back to Helium and tried a better-guess aperture shape. The match was significantly worse, with the shelf moving down the spectrum to form a separate peak due to the additional thickness for most of the blockage in the aperture.

The change in the aperture shape was designed to provide a smoother feed-in for ions to see if they scatter from the slight angle. They don't.

Simulation 22 used the refined aperture shape with 100 000 ions to test statistical accuracy. Reduced-energy fraction 0.2% of 15077 transmitted ions.

6.5 Varied-parameter simulations

Simulation 23 plots beam-width and scattered fraction of transmitted ions (for a Simulation 17 aperture of Pt, 500keV He). The point at which this graph stabilizes is the optimal beam width for these parameters. For 10000 inital particles, 900 particles are transmitted at the widest setting (1450nm) leading to reasonable statistical accuracy. Once the beam width reaches 300nm the result is stable, so the 500nm used for the above simulations (except for low-energy phosphorus, where you are safe as long as you cover 20nm either side of the aperture) is quite satisfactory.

Simulation 24 repeated this simulation for a silicon aperture. A 500nm beam width is the stable point.

Simulation 25 used a $(100\text{nm})^3$ block of solid silicon to compare range and radial-range with SRIM.

Later simulations (26 – 110) combined silicon and platinum apertures, using both materials in each simulation.

Simulation 91 ended up being the best match (calculated by a rough approximation of the Chi-Square difference testing algorithm).

6.6 Diamond Masked Implant simulations

I altered DetectorConstruction.cc to simulate implanting MeV He-4 into diamond past a cleaved silicon mask for Julius.

6.7 Aluminium Oxide Nanotemplate simulations

I altered DetectorConstruction.cc to simulate electrons, 14keV P-31 and various He-4 energies through a porous aluminium oxide membrane for Jinghua and Paul.

6.8 Silicon nanowire implantation density

I altered the program to simulate implanting 14keV P-31 into a cylindrical nanowire.

References

[Gorelick et al., 2009] Gorelick, S., Sajavaara, T., and Whitlow, H. (2009). Aperture-edge scattering in mev ion-beam lithography. ii. scattering from a rectangular aperture. *Journal of Vacuum Science & Technology B: Microelectronics and Nanometer Structures*, 27:1109.

[Mendenhall and Weller, 2005] Mendenhall, M. and Weller, R. (2005). An algorithm for computing screened coulomb scattering in geant4. *Nuclear Inst. and Methods in Physics Research, B*, 227(3):420–430.

[Paul and Schinner, 2003] Paul, H. and Schinner, A. (2003). Judging the reliability of stopping power tables and programs for heavy ions. *Nuclear Instruments and Methods in Physics Research Section B: Beam Interactions with Materials and Atoms*, 209:252 – 258. Fifth International Symposium on Swift Heavy Ions in Matter.