

Simulating VITO

Alex Fragapane

Supervised by: Monika Stachura and Alex Gottberg

1 Introduction/Motivation

Currently, plans are in place for the ASPIC (Apparatus for Surface Physics and Interfaces at CERN) beamline at ISOLDE to be upgraded to the VITO (Versatile Ion-polarized Techniques On-line) line. The ASPIC line performed solid state physics research of magnetism at surfaces and interfaces. It used a UHV (ultra high vacuum) environment and worked on samples produced by MBE (molecular beam epitaxy). The beamline was operational for about a decade, but has been out of use since 2006. Recently there has been a push to restart this beamline. Additionally, there has been a push to do β -NMR research at ISOLDE. The VITO line is a compromise between these ideas which seeks to upgrade the existing ASPIC line to be suitable for β -NMR. Because most of the components from ASPIC will be reused, the new line is a cost effective method to be able to continue ASPIC research while granting the ability to do β -NMR. Additionally, there will be a third end station left open for traveling experiments. The major upgrade is a laser based spin polarization tube. This upgrade will provide the capability to produce spin-polarized nuclei necessary for β -NMR and other nuclear physics experiments. To implement this beamline we have been simulating the new geometry in order to fully characterize and optimize the line. This paper will discuss the techniques used to simulate the pieces and provide a brief outline for what needs to be done as a continuation of this work.

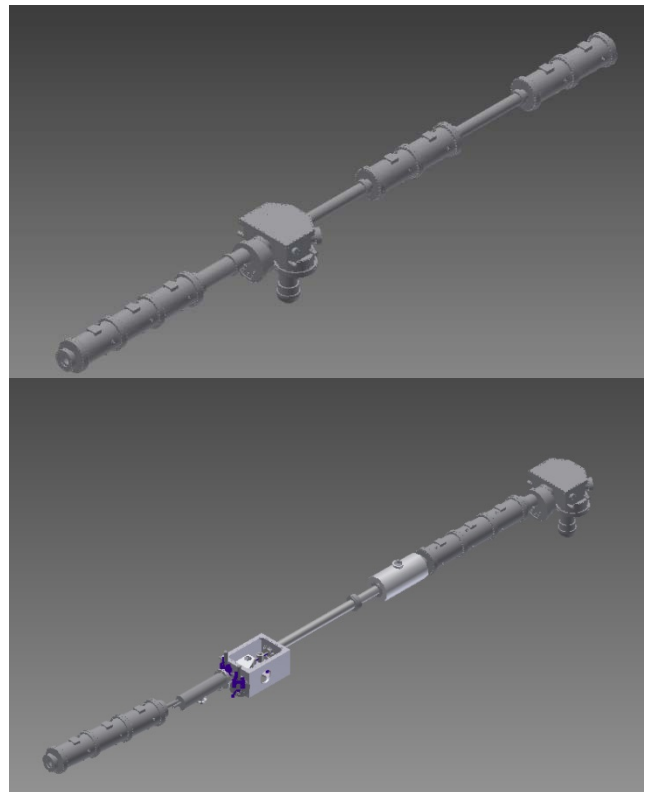


Fig 1.1 Top: ASPIC Bot: VITO

1.1 SIMION

The program chosen to simulate VITO is SIMION version 8.1. SIMION is an ion optics program which calculates ion trajectories by solving Laplace's equation for a given set of electrodes. Because VITO uses a non-standard geometry, SIMION's ability to implement completely user-defined geometries has been essential for simulating the beamline. The user has the option to upload geometries from CAD programs or define their own geometries through the use of GEM files. Additionally, the user has the ability to completely define the initial ion distribution, adjust electrode potential, as well as create adjustable variables through LUA user programs. These features will be discussed in greater detail in the coming sections.

2 The Process

Here, I will go into detail about the process I have used to simulate a part on SIMION. This involves taking a piece from Autodesk Inventor and then running simulations and taking data in SIMION. It is important to note that while this has been my process, there are other methods of doing some of these tasks which I am less familiar with.

2.1 Preparing a Piece

When preparing a piece to be analyzed by SIMION, we begin with a CAD drawing in Autodesk Inventor. From here, we need to import this piece into SIMION using the SL Tools function. SIMION will only accept CAD files of type STL, so the drawing from Inventor needs to be exported as an STL file. In Inventor, we select "Export->CAD Format" from the File menu and then select STL as the file type. SIMION will read each individual STL file as a single electrode. Because of

this, we must ensure that every electrode in our CAD drawing is accounted for as an individual STL file. When selecting the file type, we must click the options button and change the settings to export the assembly as “One file per part instance”. This will create a unique STL file for each piece in the assembly. However, this will cause an excess of STL files as we do not want each screw to be considered by SIMION as an electrode. To fix this, we create a single piece for all parts which will be on ground and export this file separately from the electrodes. This way we will be left with one STL file for everything on ground and one STL file for each electrode.

To begin this process we must first create a copy of the original CAD drawing. To make a copy, we select Copy from the Pattern tab on the Assemble menu. From there we are given the option to select the piece we wish to copy, what the file suffix will be, and where to save the file. It is important to select the option to open the copy in a new window. If this is not selected, the copy will appear in the current assembly and we cannot save it as a separate file.

Now that we have a copy of the piece, we can create a single part out of everything except the electrodes. In Inventor, create a new standard part. Under the Insert tab in the Manage menu, select “Derive”. Select the copy of the assembly we have just made. Next, you will have the option to include or exclude each piece in the assembly. We wish to exclude all of the electrodes from this part so that only pieces which will be on ground are left. It is also important to note that insulating materials close to the beam should also be excluded as having them on ground can lead to false results in the simulation. Once this is done we now have a single part containing all of the grounded components of our piece. Now we just save this and export it as a single STL file, as above.

From here, all that is left is to export the electrodes. In our copy of the drawing, delete all pieces which are not electrodes. Save this and export to STL. Now, we should be left with one STL file for each electrode and a single STL file for the body of our piece.

2.2 Importing a Piece

Once we have all of the necessary STL files, we need to name them according to SIMION’s naming convention. When converting multiple STL files to be part of the same potential array, SIMION requires that they all be in the same directory and be named like so: “Piece-3.stl”. Here, every STL file should be titled Piece-, and numbered one through the number of files. This is because when giving SIMION the file location, it must be in the form: “C:\Users\YourFolder\PieceFolder\Piece-%.stl”. The “%” symbol is SIMION’s cue that there are multiple electrodes in this potential array and to scan in each number.

To perform the import, we open SL Tools from SIMION’s home screen and then select STL->PA. In the input file box, we put in our file path as above. There are many options on this screen, but I have only found use for two of them. First, the region rotate choice allows the piece to be converted in any orientation you wish. Secondly, the region scale option allows us to control the number of grid points SIMION will use. The number of grid points SIMION uses corresponds directly to the resolution of your piece. The maximum resolution in SIMION 8.1 is 20 billion grid points. However, this will produce a file of size 190GB for EACH individual electrode in the array. So, it is important to use the resolution necessary for the piece while understanding that additional grid points will quickly consume hard disk space. Tips for dealing with this memory issue will be discussed in the full beamline section.

Once the scale and orientation is decided, all that is left is to convert. This will produce a PA# file which needs to be refined. A PA# file is what SIMION calls a “fast adjustable” potential array file. This means that, once refined, the user has the ability to change the potential on any of the electrodes. Refining the PA# file is when SIMION calculates Laplace’s equation for each grid point in the array. The file must be refined before we can perform simulations. To refine the file, we load our PA# file by clicking Load on the SIMION home screen and selecting our file. Then we select the refine option on the home screen and click “Refine” to start the process. This can take anywhere from a few seconds to half a day depending on the number of electrodes and the number of grid points used. Once this is completed, we are left with a PA0 file and PA files numbered 1 through the

number of electrodes. The PA0 file is the file we must load into SIMION to run the simulations. The numbered files are necessary to keep as the PA0 file references them. With this in mind, we are now ready to run a simulation.

2.3 Simulating a Piece

To begin a simulation, we load our PA0 file in SIMION and then click View/Load Workbench. From here we must first define the size of our work space and our initial ion distribution. The workspace size is defined in the workbench section. Be sure that it is large enough to see beam features such as foci. Note that no fields are calculated outside of the original size of the array. The scale of our piece in the workbench will typically be different from that of the actual piece. So, we must remember to either enter a scaling factor in the PA tab or scale our results manually.

The ion distribution is defined in the particles tab. For VITO simulations, we typically work with ions of kinetic energy between 30 and 50 keV. To begin with, we use an initial distribution which is a 3D Gaussian and a particle mass above 25 amu or so. In SIMION 8.1, the number of particles is limited to 1000. We can force the program to increase this cap, but using a realistic number of particles is very RAM intensive. Initially, these are sufficient beam conditions to ensure that under ideal circumstances, the piece behaves as expected. However, it is necessary to use a less ideal beam to test the limits of VITO's geometry.

To shoot particles through the beam, click "Fly'm". Once we are sure that our beam works with no applied voltages, we can change the voltage of each electrode using the fast adjust tool under the PA tab. If there are obstructions or irregularities in the beam, it is best to view these using the Z3D tool. This tool will make cuts which allow you to view inside of the piece. If a Z3D cut is made in the XY view, the inside of the piece can be viewed from the XZ, ZY, and 3D Iso views. Another tool for viewing problems with the potential array is the PE view under the PE/Contours tab. This view gives a graphical representation of the potential energy along the array. Note that the electrode must be visible for this view to work so you may need to cut into your beamline before using PE view.

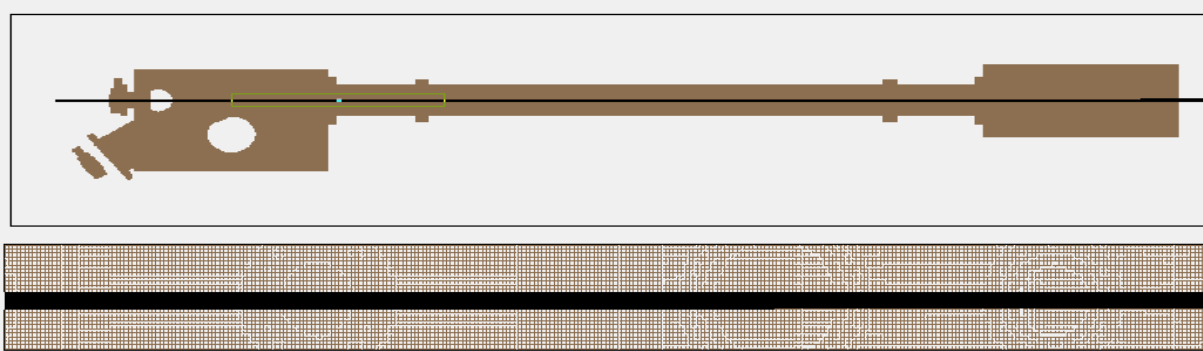


Fig 2.3.1 Top: XY View, Green rectangle where Z3D cut will be made. Bot: XY View after cut is made

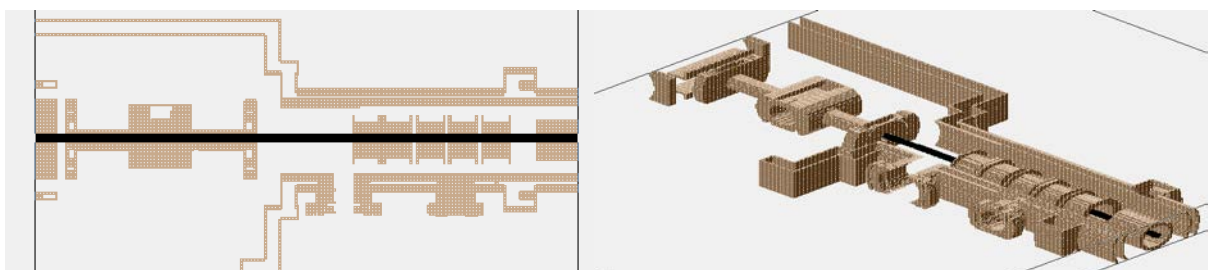


Fig 2.3.2 Left: XZ view after cut. Right: Isometric view after cut.

To take data in SIMION, we can use the data recording option under the Particles tab. We can choose what data to take, where to take it, and what format to output the data in. What is necessary is completely dependent on the simulation. I will go into more detail on a few specific

methods I have used to take and use data when I discuss pieces I have simulated. This is a very, very brief overview of what is possible in a SIMION simulation. It is difficult to go into more detail because what is necessary is situation-dependent. Because of this, I will explain more explicitly methods of simulation and data analysis I have used when discussing each piece I have simulated. For more information on these topics, I refer you to the SIMION manual, which provides an in-depth discussion on all of SIMION's capabilities. Furthermore, I refer you to the example and courses folders in the SIMION directory; they are very helpful in learning the full breadth of SIMION's functionality.

3 VITO Parts

3.1 Single Electrostatic Quadrupole

VITO's current geometry contains two sets of three quadrupoles. The purpose of these is to focus the beam. So, as a first step, I simulated a single quadrupole to get an idea of its focal length as a function of voltage. Finding the focus of this piece involved three major steps; setting user variables through user programming in SIMION, running SIMION in non-gui batch mode, and analyzing output data.

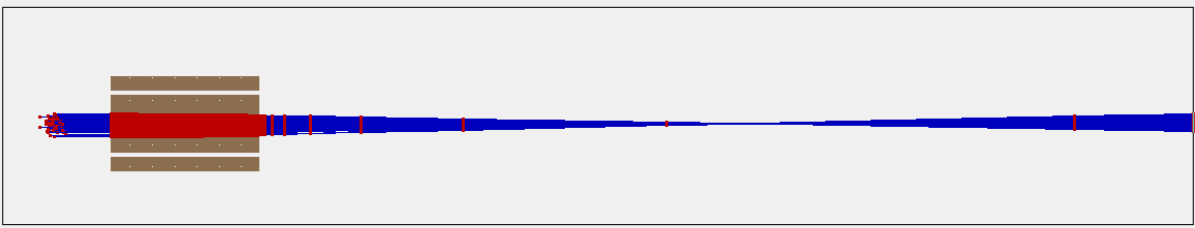


Fig 3.1.1 Electrostatic Quadrupole (view from focal plane)

3.1.1 User Programs

SIMION supports user programs written in lua. The program file must have the same name as the workbench file and will be called automatically each time ions are flown. If our workbench is titled Quadrupole.iob, the associated user program is called Quadrupole.lua. These programs give the user the power to define their own variables which can be changed in the workbench. Besides this they give the user the capability to define an ion distribution, when to return data, create alternating potentials, and many other things. I cannot go into too much detail as I have only used a small fraction of their capabilities. I will only be able to explain how I have implemented these programs. A much more thorough explanation of user programs can be found in the SIMION manual (Appendix L).

For the quadrupole, the user program was used to be able to adjust the potentials as a single variable. In the ideal case, opposite ends of the quadrupole will be at the same potential and the other two electrodes would be at minus that potential. So, the lua program was used to set a user defined variable which would set all four potentials in this way. For good examples of lua code, I highly recommend viewing the examples folder which is in the SIMION program folder.

3.1.2 Batch Mode

For simulations in which we wish to scan a series of potentials in a systematic search, it can be tedious to change the values in the graphical interface. In cases such as this one, it is useful to

run SIMION in batch mode. To do this, we need to be familiar with SIMION's command line interface. This way, we can simply write a program to feed commands to the Window's command line which instruct SIMION to run simulations and take data. A list of all the commands can be found in Appendix M of the SIMION manual.

In this specific case, I needed to adjust my user defined variable corresponding to the voltage of the quadrupole. I wrote a program in C++ which would tell SIMION to run a simulation at some voltage, increment that voltage, and then repeat the simulation until the maximum voltage had been reached. The command fed to SIMION looks like this:

```
simion --nogui fly --adjustable n= currentVoltage --grouped=1 --recording-output=
outfilename C:\Users\afragapa\QuadrupoleFolder\Quadrupole.iob
```

The command `--nogui fly` tells SIMION to run a simulation outside of its graphical interface. `--adjustable n=currentVoltage` tells SIMION to set the user defined variable, `n` to some voltage. `--grouped=1` tells SIMION to fly the ions at the same time rather than individually. This is important for how the data is formatted. If the ions are grouped, the outputfile will have data for each ion at every time step. If the ions are not grouped, all of the data for a single ion will be given before any data for the next ion is given. `--recording-output=outfilename` tells SIMION to record data and to output the data to a given filename. Lastly, the path of my workbench tells SIMION where to look for the simulation it is to run. By incrementing the `currentVoltage`, we can have SIMION run and take data for many different voltages at once. In the attached code, you can see how this technique was implemented.

3.1.3 Data Analysis

The data in the output files was defined through the data recording options in the SIMION workbench. The data to be returned was the Ion number, time of flight, and the x,y, and z positions. The data was to be returned at each time step. This means that every time a new ion position was calculated, it would be returned to the output file. The format of the data was delimited by tab with no header options selected. This is so the data would be in a form easy to parse through with C++. When the command is fed to SIMION by the program to record data, the options currently selected in the workbench will be used.

To find the focus, we wish to find the point at which there was the smallest difference in ion position along the focal plane. So, a program was written to find the maximum difference between ions in the focal plane at a given time step. Then, the program would compare this difference to other time steps. If this difference was smaller than the max difference in the next thousand time steps, it was determined to be the focal point. To see this explicitly, refer to the attached code. In this way, I was able to use the batch mode approach to find the focal point as a function of voltage.

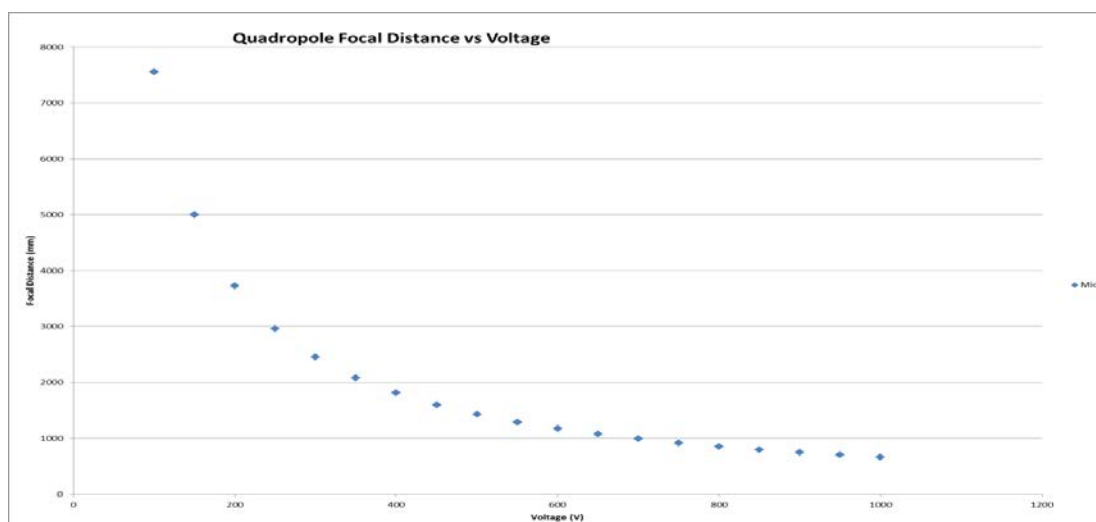


Fig 3.1.3.1 Quadrupole Focal Distance vs. Voltage

3.2 Laser Based Spin-Polarization Tube

The major upgrade from the ASPIC beamline to the VITO line is the polarization tube. The purpose of this tube is to polarize the nuclear spins of incoming ions so that β -NMR and other nuclear physics experiments can be performed. There are tube shaped electrodes inside the piece which bring ions to an energy such that the laser energy is at the correct transition energy. The task with this piece was to ensure that the ions smoothly transitioned into and out of this energy and that there were no unwanted focusing or divergence effects after the tube. The three long electrodes inside the polarization tube were set to -5000V. On the front side there are five steps down to this voltage and on the backend there is a decelerator with 52 plates. The steps as well as the decelerator plates were used as equal steps down and up from -5000V.

The best way to observe this was to shoot a single ion through and track its kinetic energy at each time step. In the output file, SIMION returned that ion number, time of flight, x position, and kinetic energy. A program was written in C++ to extract the x position and kinetic energy of the ion from the output file and write it to a csv file. To output to a csv file, columns are defined by commas and rows are defined by lines. Once again, this can be seen in the attached code.

The kinetic energy of a single ion was then graphed to observe the transition areas. As seen in figure 3.2.2, the polarization tube worked as expected. There are a couple things to note. At first, pieces which connect the electrodes inside the tube were included on ground. As seen in figure 3.2.1 (bot), this caused large fluctuations in the kinetic energy of the ion. This is because these pieces were meant to be insulating. To rectify this, one must simply be sure to exclude all insulating pieces which are close to the beam from the simulation. It is important, however, to keep in mind whether ions could potentially hit them as they will not be in the simulation at all.

The divergence effects were viewed in the graphical interface. As seen in figure 3.2.1 (top), not including the deceleration causes a steep drop in kinetic energy which produced divergence effects in the final beam.

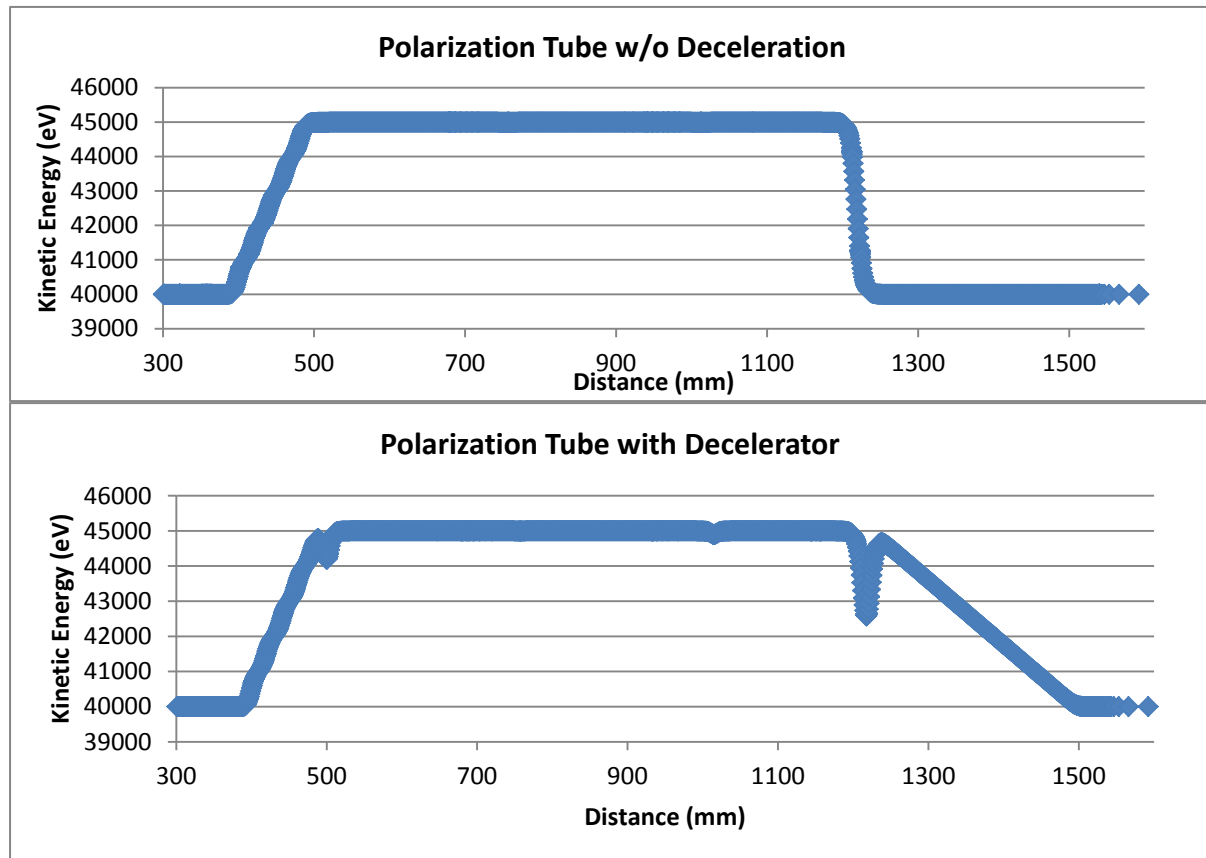


Fig 3.2.1 Top: Deceleration and insulators not included Bot: Deceleration with insulators included

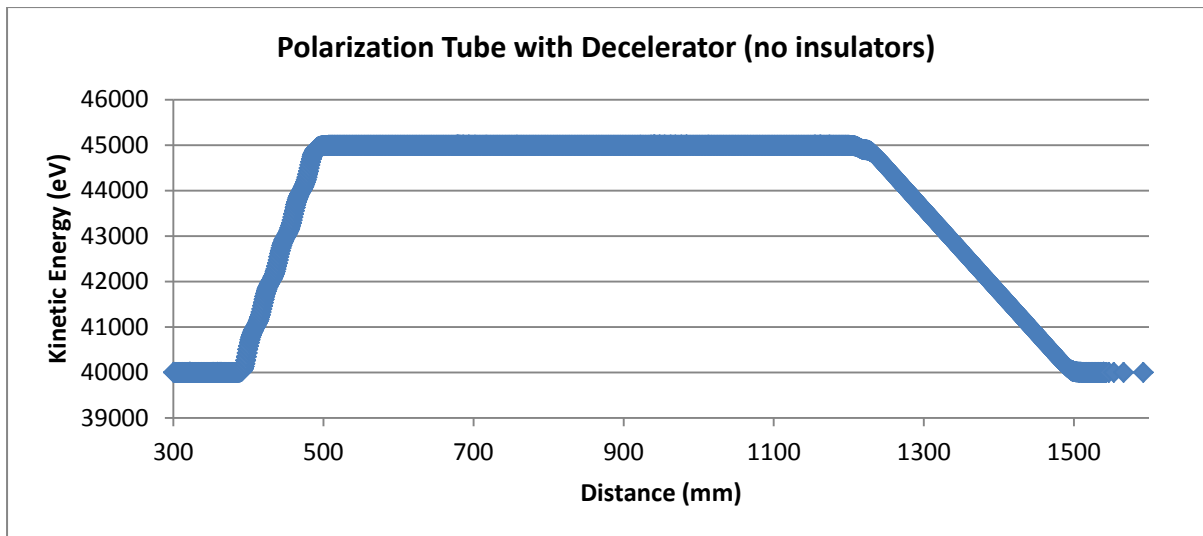


Fig 3.2.2 Decelerator included and insulators excluded

3.3 Quadrupole Triplet vs. Doublet

A brief analysis was also done on the differences between a quadrupole triplet vs. doublet. The current geometry has a triplet, followed by a five degree deflection before the polarization tube. If a triplet alone cannot properly focus the beam inside the polarization tube, an alternative is to have one doublet before the deflection and one doublet before the polarization tube. To begin determining this, the focal properties of the two pieces were briefly tested. This was also in part to learn how to focus a quadrupole triplet.

Voltages which provide a focal distance of about one meter for the quadrupole triplet with a parallel beam are 700V for the first, 1200V for the second, and 600V for the third. I have found no infallible pattern when focusing the triplet, but the difference between voltages tends to be similar to this. Furthermore, for the VITO beamline, voltages around these tend to produce the desired focal distance.

For a realistic beam, we use a beam spot of 4mm which diverges to 12mm over one meter before the triplet. To do obtain this divergence, select Angles under Direction in the particle definition screen. Then select that the particles have a Gaussian distribution of angles. The standard deviation depends on the scale of the piece, but for our pieces it has typically been around .12 degrees. Under these beam conditions, the triplet only produces a clear focus at higher voltages. Typical voltages are 1400V, 1900V, and 1100V for the first, second, and third, respectively. The triplet simulation is currently being used as a quick way to test triplet foci before putting them into the full beamline.

4 The Full Beamline

Currently, work is being done to simulate the beam as a whole. This has been a big last step because, while characterizing individual pieces is useful, finding out how they work in conjunction with each other is what determines the quality of VITO's geometry.

4.1 Memory

One of the biggest challenges with simulating the entire beamline is hard disk space. To simulate the line accurately, we need grid points at least on the order of one hundred million. This produces a file several gigabytes in size for each of VITO's 96 electrodes. If it turns out we need to go to even higher resolutions, the beam file size will exceed our capacity to store it. There are several compromises we can use to reduce file size. The most promising option is to split the beamline into two large pieces. The problem with this is that the fields calculated in one portion will

have no interaction with the fields in the other portion. The split will likely need to be between the deflector and the polarization tube. This is because between these two points there is the smallest possibility of fringe field effects. Another option is the difference between a PA and a PA# file. While a PA# file can be fast adjusted to any voltage, a PA file is limited to a single voltage. However, there is a way to couple electrodes which are linearly dependent, greatly reducing memory. Once we have a better idea of the beamline, investigating this method might be a good way to reduce disk space.

4.2 Focusing the First Triplet

For now, all the work done has been to find what voltages will focus the first quadrupole triplet and provide good transmission. These simulations have been done with a beam energy of 40keV and ion mass 50amu, using realistic beam conditions. This begins the beam with a 4mm diameter one meter before the beamline which diverges to 12mm at the beamline. To accomplish this, we have used a 3D Gaussian distribution of initial ion position with standard deviation of .3mm. Additionally, the initial velocity distribution is defined by a Gaussian distribution of angle (Elevation, Azimuth) with a standard deviation of .12 degrees. The beam in SIMION is scaled down by a factor of two and these values produce a realistic beam also scaled down by that factor. When performing this simulation, the voltages used for other components along the beam are the ones listed in Appendix B.

Ideally, the beam should be focused inside the polarization tube. However, it was found that the main challenge in doing this is at the charge exchange cell. The charge exchange cell has the smallest diameter along the entire beamline and occurs directly before the polarization tube. If the beam is focused too far along the polarization tube, ions are lost at the entrance to the charge exchange cell. If the beam is focused at the charge exchange cell, there is divergence along the polarization tube. The goal will be to optimize the geometry to provide a compromise between these two scenarios. It has been suggested that an einzel lens be placed after the charge exchange cell to refocus the beam before the polarization tube.

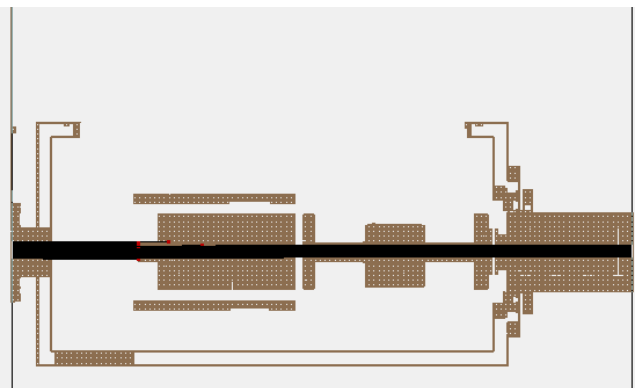


Fig 4.2.1 Ions lost at charge exchange cell

With the current geometry, the voltages which provide 100% transmission are 1200V, 1700V, 1000V for the first, second, and third quadrupoles, respectively. Additionally, this configuration produced a focus towards the beginning of the polarization tube and manageable divergence afterwards. This can be seen in the figure below. Other voltage combinations and their effects are outlined in Appendix C.

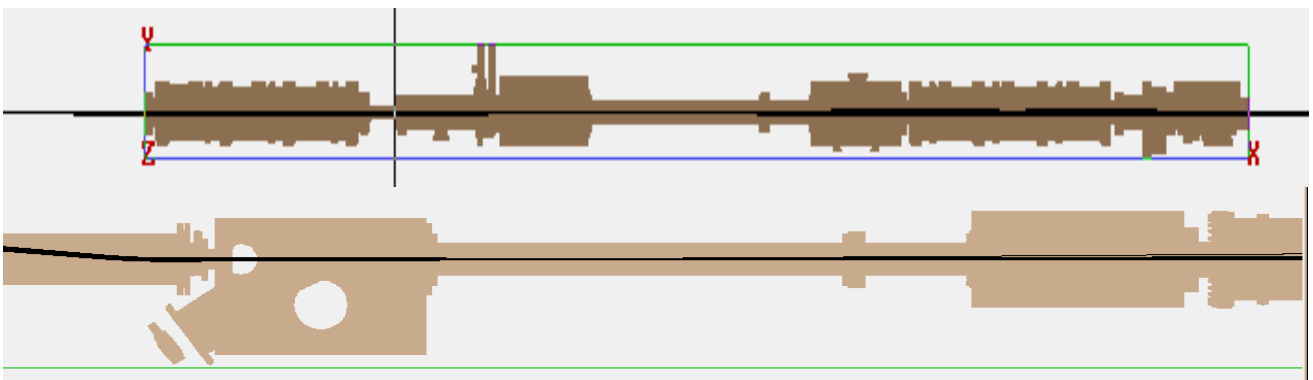


Fig 4.2.2 Top: XY view Bot: Zoom of XZ view

5 Still to do

Now that we have a working beamline simulation, the goal is really to test the limits of this geometry and then optimize it. To do this, a few things must be accomplished.

5.1 Beam Conditions

First, most of the simulations done thus far have been done with a parallel beam with initial positions forming a 3D Gaussian distribution. The standard deviation used depended on the scale of the piece, but the goal was typically to have a beam diameter of 10mm. This, of course, is a hopelessly ideal case. We would like to run simulations with realistic beam conditions and then move to even worse conditions. This way, we can really probe VITO's geometry for bottlenecks. We know that the beam emittance at the ion source is 10π mmrad. To simulate this, we have begun to create an initial beam of 4mm which diverges to 12mm over one meter. This is done using a Gaussian distribution of elevation and azimuth angle with a standard deviation of .12 degrees. From here, we need to vary the location of the 4mm spot and the standard deviation of the angles to see what the limits of initial beam condition are. Additionally, most measurements have been done at 40keV. The beam energy will be anywhere between 30 and 50keV, so a wider range of beam energy must be tested.

5.2 Switchyard

Another thing to note is that the switchyard is not yet complete. There are some design flaws which need to be fixed before we can proceed with accurately simulating it. For now, the full beam shoots through the center of the switchyard or the right deflector. It is important that the switchyard be added accurately into the simulation as the center will be the least important exit.

5.3 Data

First, we would like to get an idea of beam transmission under different beam conditions. It is important that we get nearly 100% of ions through the beam, so testing the limits of the geometry in this sense is pertinent. This is possible simply through telling SIMION to return position and event data at the ion's "splat" (termination). This will tell you whether the ion hit an electrode or made it outside the workbench. Furthermore, by observing the x position at this time we can see which parts of the beam cause the most problems.

Secondly, we would like to get an idea of the size of the focus we can produce after the beamline and the divergence of the beam afterwards. Ideally, both the focus and beam divergence will be as small as possible. There will be three pinholes before the experiment which the beam must be able to go through. One way of testing that the focal point is good enough is to create these three small pieces in Autodesk Inventor and include them as part of the beam simulation. This way we can explicitly see whether the beam will make it through the holes.

Lastly, calculating the emittance at certain points in the beam might be valuable. There is an example user program in the SIMION examples folder which calculates the emittance. This might be modified for our purposes. It is worth noting that there are many useful lua programs in the SIMION examples folder which might prove useful.

5.4 Optimization

The ultimate goal of these simulations will be to optimize VITO's geometry. To do this, we must identify the bottlenecks in the beam. This means we need to know where ions are being lost or where beam conditions tend to become suboptimal. This will inform where to put beam diagnostics in the actual beamline.

Additionally, we would like to optimize the distances between components. For instance, given proper beam conditions, it may not be possible to produce focuses exactly where we would like given the current geometry. By extending or contracting parts of the beam, we will be able to optimize for these conditions. This process may be possible through lua user programming, though I

expect it will be easier to perform these changes by hand. See the SIMION example file, geometry_optimization for tips on how to use lua to optimize beam voltage and geometry. They use GEM files instead of STL files for their geometries, so I doubt the process would be the same. Still, it is useful to see how this can be done and perhaps adapted to our format.

Lastly, we wish to optimize the number and placement of the components. For instance, discussed earlier was the question of using a single quadrupole triplet initially or two sets of doublets. These are the types of questions we can begin to explore now that we have managed to import the full beam geometry and are beginning to use realistic beam conditions. Another option is to put in small einzel lenses along the beam if there are bottlenecks where the quadrupoles cannot focus properly. All of this must be determined in the coming weeks with the continued simulation of the VITO line.

Contact Information

If there are any questions at all regarding the work I have done, SIMION, or anything at all, please feel free to contact me.

Alex Fragapane
+1 (949)-922-8712
ahf27@georgetown.edu

Appendix A -Programs

Program to run single quadrupole simulation in batch mode:

```
#include <iostream>
#include <string>
#include <fstream>
#include <stdlib.h>
#include <sstream>

using namespace std;

int main () {

    /* Variable declarations */
    string projName;
    int min = 0;
    int max = 0;

    int step = 0;
    string outfilename;
    // Tracks the number of fly'ns
    // which helps parse the output file
    int flyCount = 0;
    // Will hold the commands to pass to system()
    string command;
    //Fully resolved path to append:
    string AlexPath = "C:\\Users\\afragapa\\SimionThings\\QuadropoleStuff\\";
    /* End variable declarations */

    cout << "Enter name of project (case-sensitive, no file extensions): ";
    cin >> projName;

    cout << "Enter the minimum voltage to try for the electrodes: ";
    cin >> min;
    cout << "Enter the maximum voltage to try for the electrodes: ";
    cin >> max;

    cout << "Enter the step size to adjust the voltages: ";
    cin >> step;
    cout << "Enter the output file name: ";
    cin >> outfilename;

    for(int currentV=min; currentV<=max; currentV+=step) {
        /*
        cast the int as a string to concatenate. If you can compile C++11, to_string() from the std library
        should work just as well. Here is the to_string() version of the next line:
        command = "simion --nogui fly --adjustable n_volts=" + std::to_string(currentV) + " --recording-
        output=batchtest.txt" + AlexPath + projName.c_str() + ".IOB";
        */
    }
```

```
stringstream temp;
temp << currentV;
string castV = temp.str();

// Update command to hold the fly instructions.
// You may want to adjust more parameters later, such as grouping
// Later change this to user-defined output path.
command = "simion --nogui fly --adjustable V=" + castV + " --grouped=1 --recording-output=" +
outfilename + " " + AlexPath + projName + ".iob";
// Now fly'm!
system(command.c_str());
// Increment the count of the number of fly'ms and the curret voltages.
flyCount++;

}

return 0;
}
```

This program parses through the output data given by the quadrupole batchmode program. It expects data taken at every time step delimited by tab in the form: Ion number, time of flight, x, y, z.

```
#include<iostream>
#include<fstream>
#include<string>

using namespace std;

//MUST CHANGE THESE IN CODE IF CHANGED IN SIMION SETTINGS
const int NUM_OF_IONS = 100;
const int TOLERANCE = 1000;

int main() {

    ifstream dataFile;

    int gapCounter = TOLERANCE;
    float xVals[NUM_OF_IONS];
    float yVals[NUM_OF_IONS];
    float zVals[NUM_OF_IONS];
    float xFocalY[NUM_OF_IONS];
    float xFocalZ[NUM_OF_IONS];
    float yFocal[NUM_OF_IONS];
    float zFocal[NUM_OF_IONS];
    float currentMinY;
    float currentMaxY;
    float currentMinZ;
    float currentMaxZ;
    float minGapY = 99999;
    float minGapZ = 99999;
    int currentION=0;
    string trash;
    string fileName;
    string outfileName;

    cout << "Enter the name of the output file to analyze (.txt): ";
    cin >> fileName;

    cout<< "Enter the name you wish to output the results to (.csv): ";
    cin >> outfileName;
    ofstream outFile(outfileName.c_str());

    // Open the data file
    // File may need to be in the same directory as this program
```

```

dataFile.open(fileName.c_str());

// Throw out the first line
getline(dataFile, trash);

//Will need to do a loop to account for multiple runs

while(gapCounter > 0) {

/* Set these counters to something we can guarantee they will be replaced with on the first
comparison. */
currentMinY = 99999;
currentMaxY = -99999;

// Pretty sure it throws out tabs and newlines
//First we do this for Y
for(int i=0; i<NUM_OF_IONS; i++) {
dataFile >> trash;
dataFile >> trash;
dataFile >> xVals[i];
dataFile >> yVals[i];
dataFile >> zVals[i];
if (yVals[i] > currentMaxY) currentMaxY = yVals[i];
else if (yVals[i] < currentMinY) currentMinY = yVals[i];
} //end for

if (minGapY < (currentMaxY - currentMinY))
gapCounter--;
else {
minGapY = currentMaxY - currentMinY;
gapCounter = TOLERANCE;
// Store the current new focal values
for(int i=0; i<NUM_OF_IONS; i++) {
xFocalY[i]=xVals[i];
} //end for
for(int i=0; i<NUM_OF_IONS; i++) {
yFocal[i]=yVals[i];
} //end for
} //end else

} //end while
dataFile.close();
gapCounter=TOLERANCE;

//Next we do the exact same for Z and compare

```

```

dataFile.open(fileName.c_str());

// Throw out the first line
getline(dataFile, trash);

//Will need to do a loop to account for multiple runs

while(gapCounter > 0) {

/* Set these counters to something we can guarantee they will be replaced with on the first
comparison. */
currentMinZ = 99999;
currentMaxZ = -99999;

// Pretty sure it throws out tabs and newlines
//Now for Z
for(int i=0; i<NUM_OF_IONS; i++) {
dataFile >> trash;
dataFile >> trash;
dataFile >> xVals[i];
dataFile >> yVals[i];
dataFile >> zVals[i];
if (zVals[i] > currentMaxZ) currentMaxZ = zVals[i];
else if (zVals[i] < currentMinZ) currentMinZ = zVals[i];
} //end for

if (minGapZ < (currentMaxZ - currentMinZ))
gapCounter--;
else {
minGapZ = currentMaxZ - currentMinZ;
gapCounter = TOLERANCE;
// Store the current new focal values
for(int i=0; i<NUM_OF_IONS; i++) {
xFocalZ[i]=xVals[i];
} //end for
for(int i=0; i<NUM_OF_IONS; i++) {
zFocal[i]=zVals[i];
} //end for
} //end else

} //end while
dataFile.close();

// Print results to or output file

```

```
outFile<<"yFocal, xFocalY, zFocal, xFocalZ, Min Y dist, Min Z dist"<<endl;
outFile<<yFocal[0]<<" , "<<xFocalY[0]<<" , "<<zFocal[0]<<" , "<<xFocalZ[0]<<" , "<<minGapY<<"
, "<<minGapZ<<endl;
for(int i=1; i<NUM_OF_IONS; i++){
    outFile<<yFocal[i]<<" , "<<xFocalY[i]<<" , "<<zFocal[i]<<" , "<<xFocalZ[i]<<endl;}

return 0;
} //end main()
```

This program takes the trajectory of a single ion at every time step and outputs its x position and kinetic energy to a csv file so that it can be graphed with excel. It expects a text file with data delimited by tab in the form: Ion number, time of flight, x, kinetic energy.

```
#include<iostream>
#include<fstream>
#include<string>
#include<vector>

using namespace std;

int main(){

    ifstream dataFile;
    string inFileName;
    string outFileName;
    string baseName;
    float currentX = 0;
    float currentKE = 0;
    vector<float> xVals;
    vector<float> KEs;
    string trash;
    int count = 0;

    cout << "Enter the file to analyze (no extensions): ";
    cin >> baseName;

    outFileName = baseName + ".csv";
    ofstream outFile(outFileName.c_str());
    outFile << "X, KE" << endl;

    inFileName = baseName + ".txt";

    dataFile.open(inFileName.c_str());
    if (!dataFile){
        cout << "File does not exist.";
        return 0;}

    getline(dataFile, trash);

    while(currentX < 5000){

        dataFile >> trash;
        dataFile >> trash;
        dataFile >> currentX;
```

```
dataFile >> currentKE;
xVals.push_back(currentX);
KEs.push_back(currentKE);

}

dataFile.close();

for(int i=0; i < xVals.size(); i++){

    outFile << xVals.at(i) << "," << KEs.at(i) << endl;

}

return 0;
}
```

Appendix B – Notes on Full Beamline

Electrode list and typical voltage notes as viewed in SIMION fast adjust for the full beamline. These will be based on a 40keV beam with an ion mass of 50amu using the realistic beam conditions described above. The voltages can be scaled for different energies or ion masses.

First Triplet (directions from ZY view):

Quadrupole 1: 21 bottom, 22 right, 23 top, 24 left

Quadrupole 2: 17 bottom, 28 right, 19 top, 20 left

Quadrupole 3: 13 bottom, 14 right, 15 top, 16 left

Most of the work done on the full beam so far has been to focus this triplet. See discussion above and in Appendix C for voltages.

Five degree Deflector:

Horizontal (viewed from XZ): 89 bottom: 700V, 88 top: -700V

Vertical (viewed from XY): 91 bottom: 0V, 90 top: 0V

Note that we expect in the ideal case that the vertical deflectors will not have to be turned on.

Polarization Tube (left to right):

Three long electrodes

27, 28, 87

These electrodes have been kept at -5keV. It will be necessary to vary this to about 10keV.

Step down

33, 30, 29, 31, 32

These electrodes step down in equal parts to the voltage on the three longer electrodes.

Decelerator

35, 36, 41, 37, 38, 42, 43, 46, 44, 45, 47, 48, 51, 49, 50, 52, 53, 56, 54, 55, 57, 58, 61, 59, 60, 62, 63, 66, 64, 65, 67, 68, 71, 69, 70, 72, 73, 76, 74, 75, 77, 78, 81, 79, 80, 82, 83, 86, 84, 85, 39, 40, 34

These electrodes step back up to ground in equal parts. This has been the test case, but it will be necessary to experiment with different step down curves. For instance, an exponential step down might be better than a linear one.

Second Triplet (directions from ZY view):

Quadrupole 1: 1 bottom, 2 left, 3 top, 4 right

Quadrupole 2: 5 bottom, 6 left, 7 top, 8 right

Quadrupole 3: 9 bottom, 10 left, 11, top, 12 right

This triplet has not been tested yet, so the voltages used were simply to get a beam through. The goal is to have a focus one meter after the switchyard. Typical voltages are 800, 1400, 800 for testing.

Switchyard

Kicker (viewed from XZ): 93 bottom: -1500V 92 top: 1500V

Right Deflector (viewed from XZ): 95 bottom: -1750V 94 top: 1750V

Left Deflector omitted until fix.

Unused Electrodes

24/25 in the CEC chamber

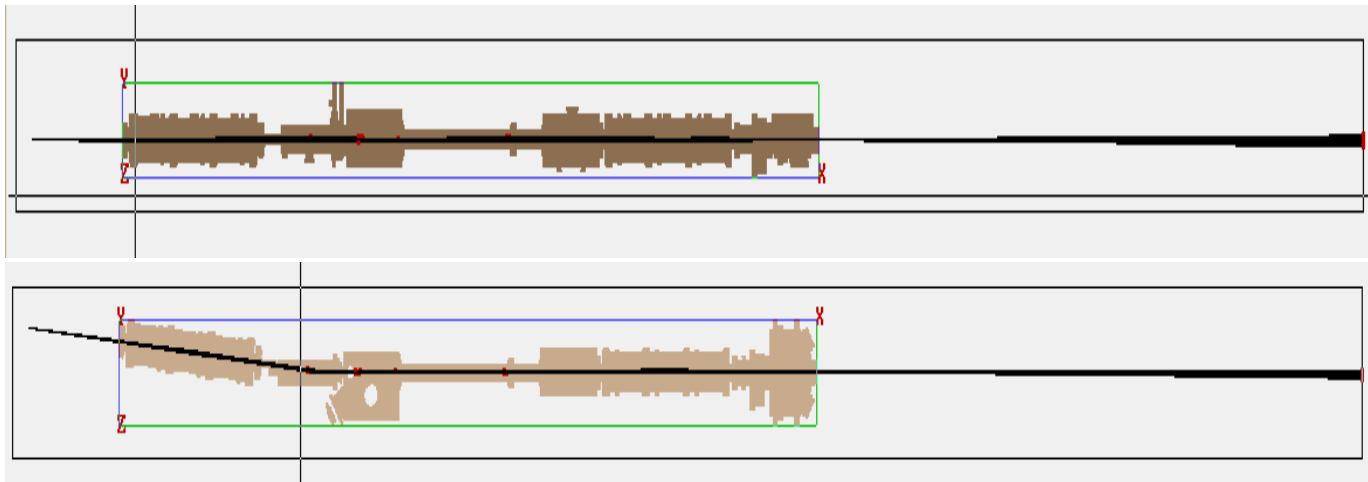
96 is the grounded body of the piece

Appendix C- Full Beamline Triplet Voltages

Voltages: 700V, 1200V, 500V

Transmission: 70%

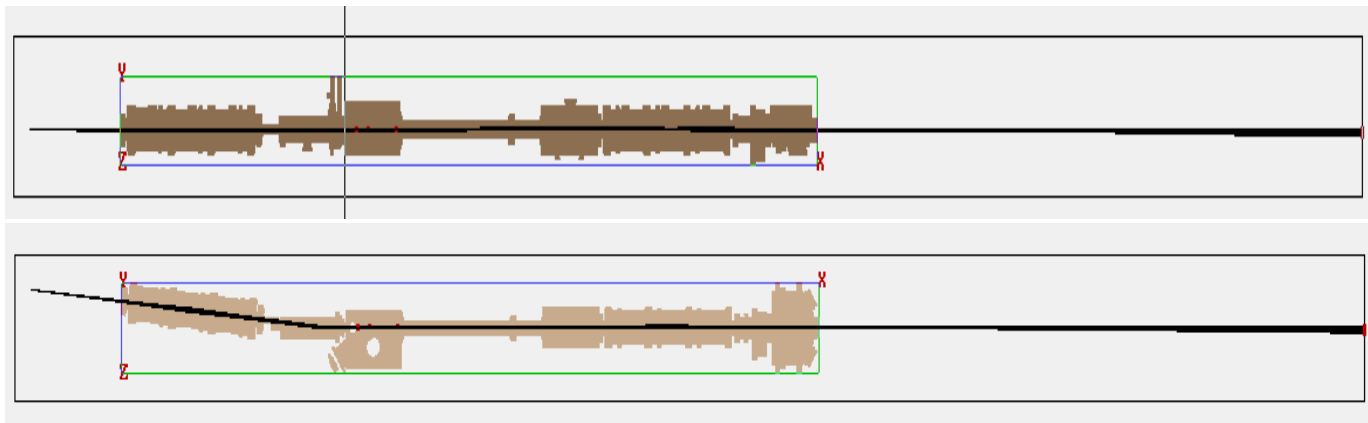
Notes: Most ions lost at charge exchange cell. With realistic beam conditions, voltages below this are not strong enough to properly focus the beam, so this was the lowest tested.



Voltages: 1000V, 1500V, 800V

Transmission: 96%

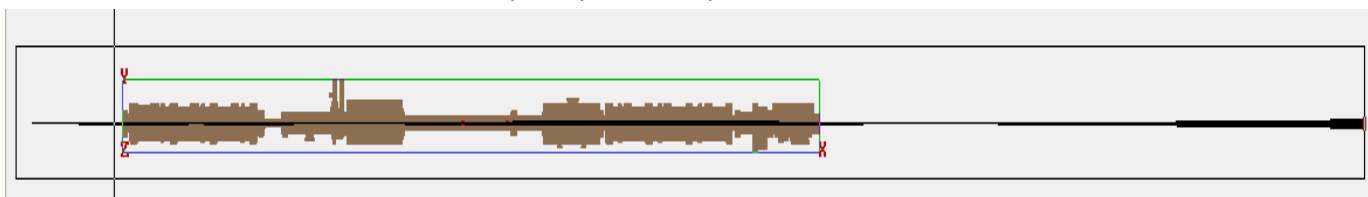
Notes: Good transmission, but all ions which were lost were lost at the charge exchange cell. The focus is less well defined, but produces little divergence which may be ideal for the laser.

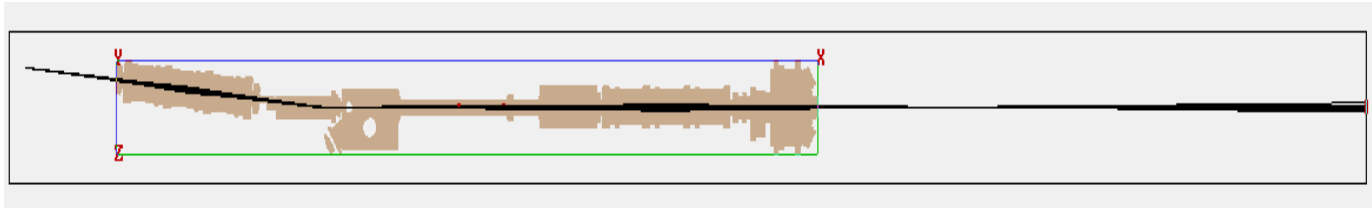


Voltages: 1400V, 1900V, 1100V

Transmission: 98%

Notes: Very high transmission, this is due largely to the focus being before the charge exchange cell, so no ions are lost. However, there is quite a bit of divergence along the polarization tube, where a few ions are lost. In this case, the focus is perhaps too early.

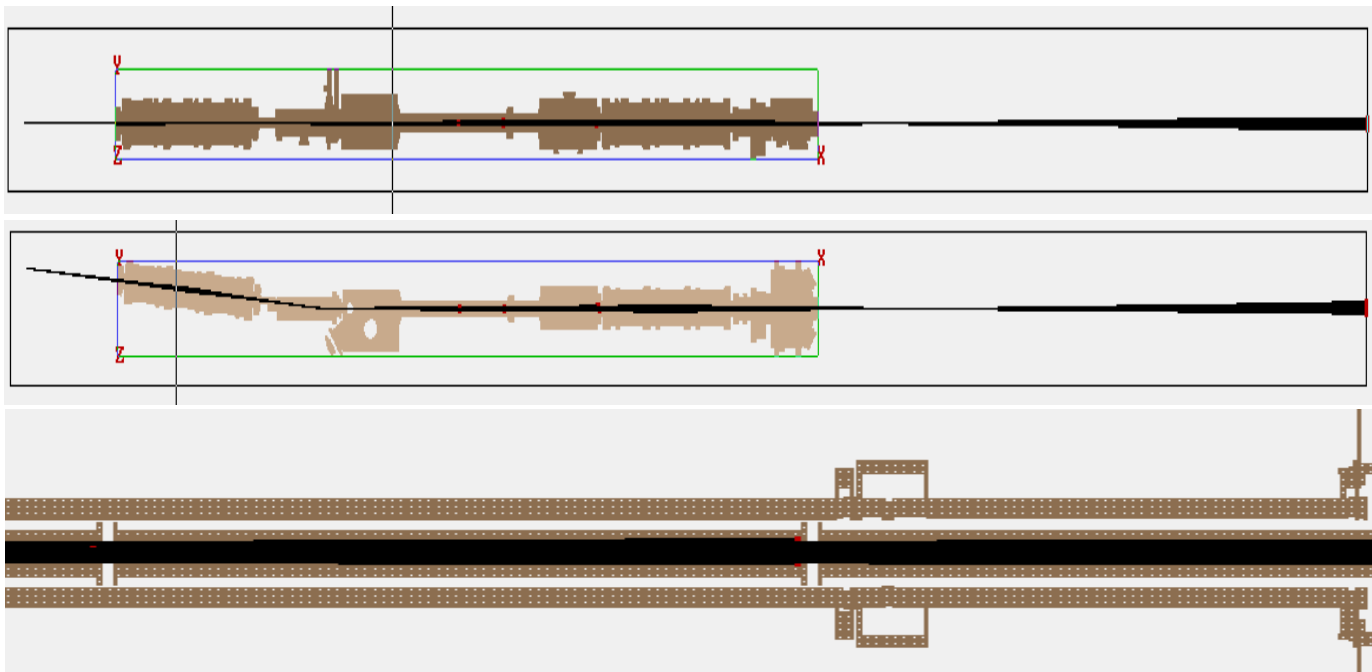




Voltages: 1600V, 2100V, 1400V

Transmission: 87%

Notes: Here, the focus was once again before the charge exchange cell, and so the ions made it past this point. However, this produces too large a divergence along the polarization tube and causes ions to hit the walls (pictured below). Voltages above this produce a focus even closer to the triplet and exacerbate this problem.



This is a very first step in tuning and optimizing this beam. This work was meant to give a good idea for an initial focus and to help determine more extensively the best values of voltage and where geometries might be changed.

References

Autodesk WikiHelp, <<http://wikihelp.autodesk.com>>

Dahl, David A., and David J. Manura. *SIMION 8.0/8.1 Manual*. 2nd ed.

Deicher, M., Stachura, M., et al., Letter of Interest to the ISOLDE and Neutron Time-of-Flight Committee. (May 2013).