ICP-MS Skimmer Cone Shock Simulations Compared to Experimental Measurements

Michael Carlson

A senior thesis submitted to the faculty of
Brigham Young University
in partial fulfillment of the requirements for the degree of

Bachelor of Science

Ross Spencer, Advisor

Department of Physics and Astronomy

Brigham Young University

ABSTRACT

ICP-MS Skimmer Cone Shock Simulations Compared to Experimental Measurements

Michael Carlson
Department of Physics and Astronomy, BYU
Bachelor of Science

The shock structure near the skimmer cone of the plasma/vacuum interface of an Inductively Coupled Plasma-Mass Spectrometer (ICP-MS) is computationally modeled using a Direct Simulation Monte Carlo (DSMC) code, FENIX. These shocks are caused by a hypersonic, rarefied gas flow hitting a metal surface. To determine the most accurate simulation of the shocks, three different gas–surface interaction models are tested against existing experimental data. The three interaction models used in this study are the specular, thermal and Cercignani-Lampis-Lord (CLL) models. The specular and thermal models are simpler to implement, but do not result in the correct shock structure. Namely, the specular model conserves too much energy in the reflected particles and does not scatter the particles enough. The thermal model does not conserve enough energy and scatters particles too much. The CLL model requires more time to set up, but results in a more accurate representation of the shock. This additional set up time comes from accommodation coefficients in the CLL model that can be set to approximately represent any surface, with its accompanying roughness and temperature. To find these accommodation coefficients, the simulated shocks need to be matched with experimental data. We found that the specular gas–surface interaction model gave the most accurate shock structure.

Keywords: ICP-MS, Gas-surface interaction, Skimmer cone, Shock formation, Supersonic expansion

Acknowledgements

I would like to thank my wife, for listening to my ramblings about my research. Also, Dr. Spencer for giving me an opportunity to do physics research. And lastly, to BYU's department of Physics and Astronomy for funding that research.

Contents

List of Figures

Chapter 1

Introduction

1.1 Description of ICP-MS

Spectrometry is the measurement of a spectrum, whether it be a spectrum of light or of mass. Mass spectrometry (MS) is a method of measuring the atomic composition of a sample by measuring the mass-to-charge ratio spectrum. It is widely used in many different settings, both for business and education. There are many different kinds of mass spectrometers, including: accelerator MS, gas chromatography MS and inductively coupled plasma-MS—the focus of this paper.

Inductively Coupled Plasma-Mass Spectrometers (ICP-MS) have many advantages over other mass spectrometry techniques. The three largest advantages are coverage, sensitivity, and speed. It can detect alkali and alkaline earth elements, transition and other metals, metalloids, rare earth elements, most of the halogens and some of the non-metals. It is sensitive to those materials up to one part in a trillion. In addition, running a sample only takes approximately four minutes. Another advantage that an ICP-MS has is that the total sample size per analysis is much smaller as compared to other methods, because of the small orifices of the skimmer and sampler cones.

There are five basic steps to ICP-MS, as shown in Figure 1.1. First, the sample is nebulized

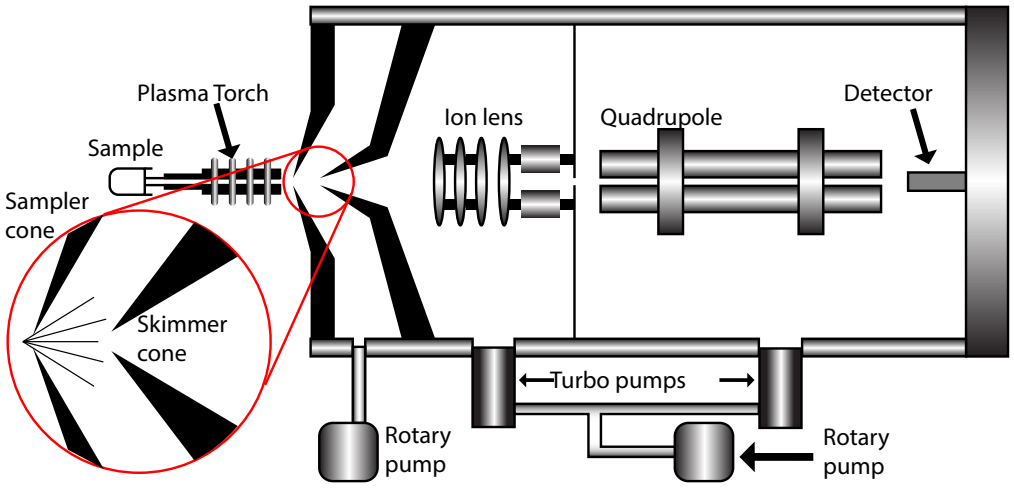

Figure 1.1 The five steps of ICP-MS are shown here: sample introduction, ionization, pressure reduction, ion focusing and data collection.

and passed through a spray chamber to ensure small droplet sizes. These small sample drops are sprayed into a stream of neutral argon gas. Second, an argon plasma torch is used to create a high temperature plasma (up to $10,000$ K). The energy of the plasma vaporizes, atomizes and ionizes the sample droplets. Third, using a pair of cones with small nozzles (<1 mm diameter), the plasma flows into a high vacuum region to reduce pressure and therefore ion–electron recombination. Fourth, the sample ions are separated from the neutral particles and focused into a beam using an ion lens. Last, a quadrupole is used to separate the particles of different masses that are then collected and recorded by their charge-to-mass ratio.

My research focuses on the pressure reduction step. There are two cones called the sampler cone and the skimmer cone in this step. The area in front of the first (sampler) cone is at atmospheric pressure. The pressure between the two cones is at about 0.001 atm. After the second (skimmer) cone, the pressure is brought down to 10^{-10} atm. Between these two cones, a shock front forms as the incoming gas beam expands outward into the low-pressure background gas. There are two pieces to this shock: the barrel shock, the surface where the particle directions are bent towards

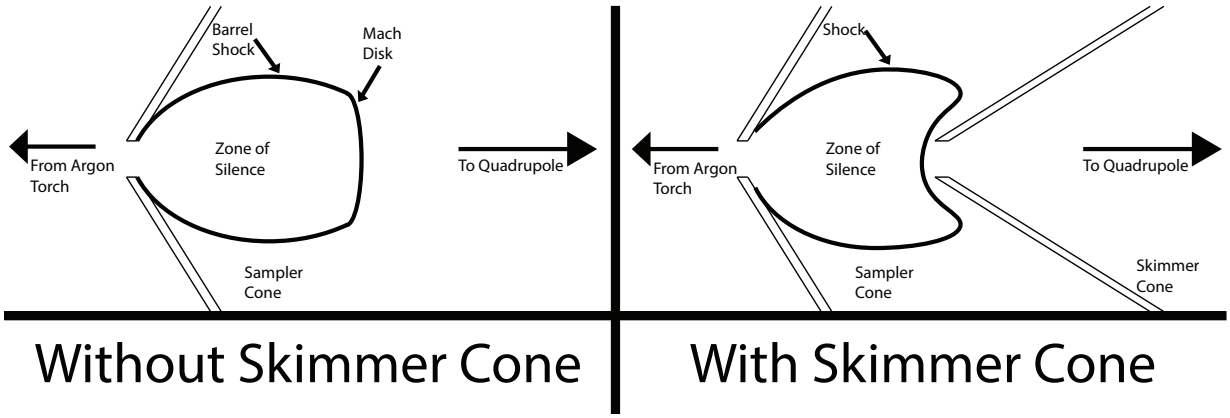

Figure 1.2 In the left figure, an idealized version of the barrel shock and Mach disk is shown. This idealized version happens when there is no skimmer cone. In reality, the shocks appear more like the image on the right, with the shock being "dragged" inwards.

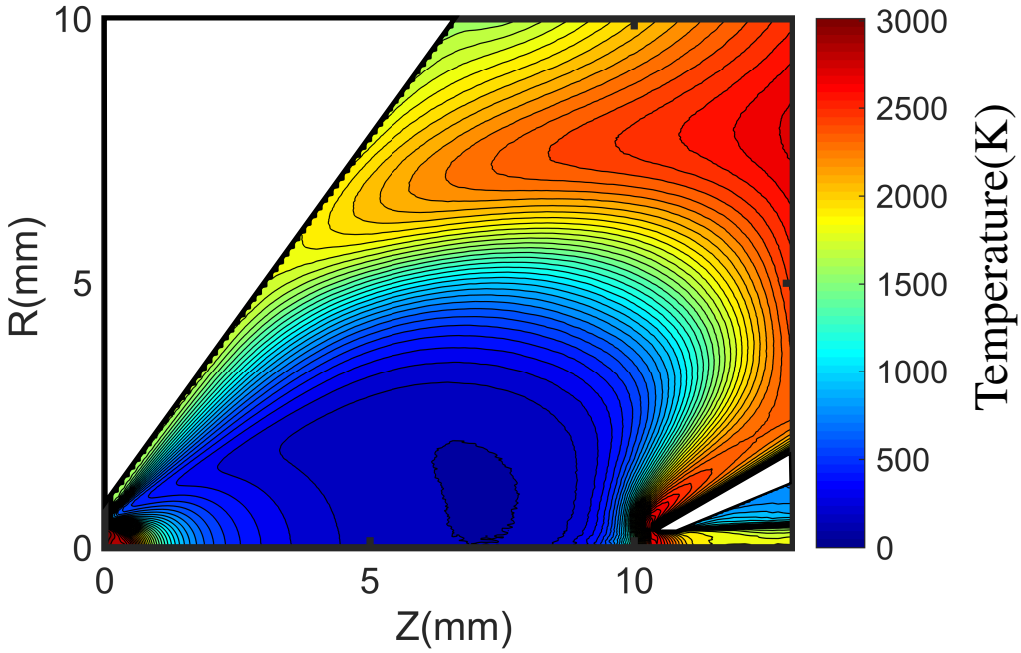

Figure 1.3 A temperature graph of what an actual shock looks like with the skimmer cone in place. The white shapes on the left and right are the cones.

the central axis; and the Mach disk, the surface where particles transform from high speed to low speed. The width of these features are approximately the mean free path, so at lower pressures the

mean free path is long compared to the size of the shock, and the features mush together to form an egg-like shape. The location and nature of these cones determine the accuracy, precision, and noise level of the measurement. If the shock terminates before the skimmer cone, significant mixing occurs with the background gas. Placing the skimmer cone inside the zone of silence is the ideal, giving just the sample gas in a confined beam. However, when the cone is placed in the zone of silence (the region where the gas is supersonic), the shock structure is modified as can been seen in schematic form in Fig.1.2. Fig.1.3 shows a simulation of what the real shock would look like. The standard barrel shock can be seen until $Z = 10$ mm, where the shock is "pulled" in the negative Z direction. The correct simulation of these shocks is the topic of this thesis.

1.2 Motivation

The ICP-MS was invented heuristically, without knowing exactly what was happening. Adjustments were made until success was reached. Conventional imaging techniques do not work on argon, the primary gas in the ICP-MS and a gas not accessible to laser-induced fluorescence (LIF). Instead, computational simulations were done. The entire ICP-MS process was simulated, from the argon torch to the ion lens, but a model for the gas interacting with the metal is needed or the simulations will not match the existing experimental data, as is shown in section 1.4.

The focus of this thesis is the shock that forms in front of the skimmer cone. To tune the shock structure, there are a few simulation parameters that can be modified so that the simulated and experimental data match more closely. Many of these parameters can be determined through other means, and are described in section 2.2. However, the gas–surface interaction model must be chosen through accuracy tests and is the subject of this thesis. The interaction model determines the reflected particle velocities as they bounce off the face of the skimmer cone.

Supersonic gas–surface interaction models are used in many fields including hydrodynamics

and aerodynamics. However, every model is an approximation to reality. It would be impractical to try to simulate a true surface, with all of its microscopic roughness. Trying to add in the actual shape of the particle would further increase the computational load on the computer. To bypass this problem, statistical models are used, randomly choosing how the particle bounces off the surface. Determining how to choose these random numbers is the crux of the matter, as including more parameters increases accuracy, but also increases computational load.

1.3 FENIX

FENIX is the code created to simulate the gas flow inside an ICP-MS. Since the vacuum interface between the sampler and skimmer cones is cylindrically symmetric, the simulation can be reduced to two dimensions (radial and axial) without any loss of generality and with a large increase in computational efficiency. Particles are introduced into the simulation through the nozzle of the sampler cone by using a region of ghost cells to ensure that the entering particles follow a Maxwell-Boltzmann distribution. Each particle is tracked as it flows through the simulated region, bouncing off the cones' surfaces.

FENIX uses DSMC [1] to statistically carry out particle-particle collisions. To reduce load, representative particles are used; each particle in the simulation represents 120 million other particles in collisions and density/temperature measurements. Instead of checking whether a particle runs into another particle by using their positions every time step—which would take billions of years to finish—particle collisions are done semi-randomly using the probabilistic soft sphere model [1]. The simulation region is split into a number of cells where particles are allowed to interact only with particles of the same cell. In each cell, an overall number of collisions is decided based on physical factors, such as temperature and density. Particles are chosen randomly to fulfill this number of collisions, taking care to not collide a particle with itself. The collisions happen as if the particles

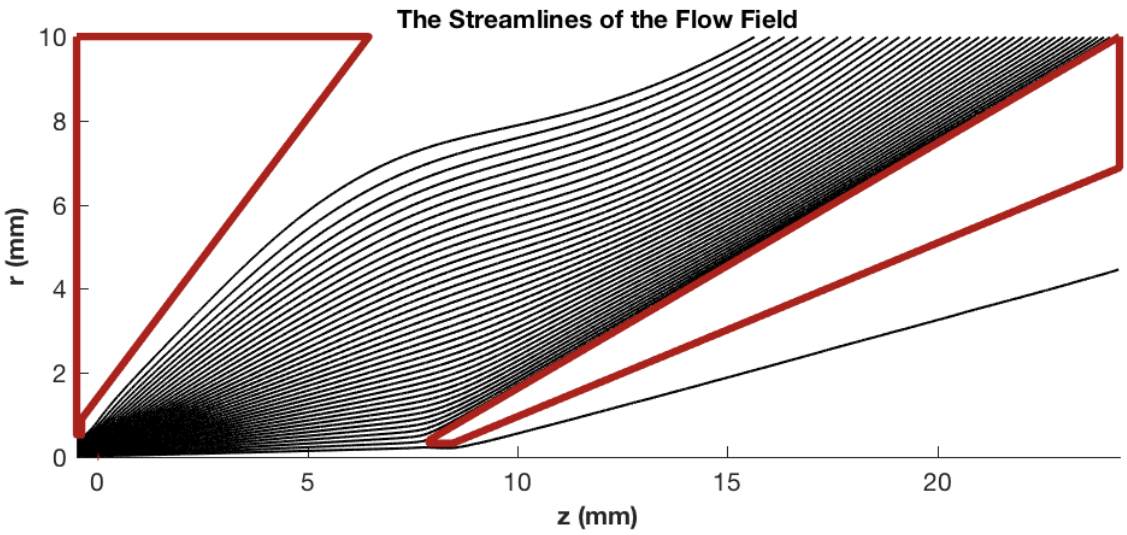

Figure 1.4 A typical FENIX simulation looks like this, where a 3-D problem is reduced to an axisymmetric 2-D problem. The particles enter the simulation from the sampler cone (left) around 1,200 m/s and expand as the streamlines show. A fraction of the particles enter the skimmer cone (right).

were next to each other with their relative velocities.

The model currently implemented in FENIX to determine the velocity vector of particles reflecting off the cones' surfaces is inaccurate and needs to be modified. Currently, a purely diffuse, thermal method is used, the details of which is described in section 2.4. As can be seen in Figure 1.5, the simulated velocity distribution doesn't match well to the experimental data. This means that the shocks are not happening in the correct places in the simulation, skewing the particle mixing that should occur. This is indicative of an incorrect model for gas–surface collisions. In Figure 1.5, specific attention should drawn to the width of the main peak—which is fixed in section 2.1 and section 2.2—and the behavior of the tail on the left-hand side of the peak, the primary metric by which the accuracy of the various models is measured.

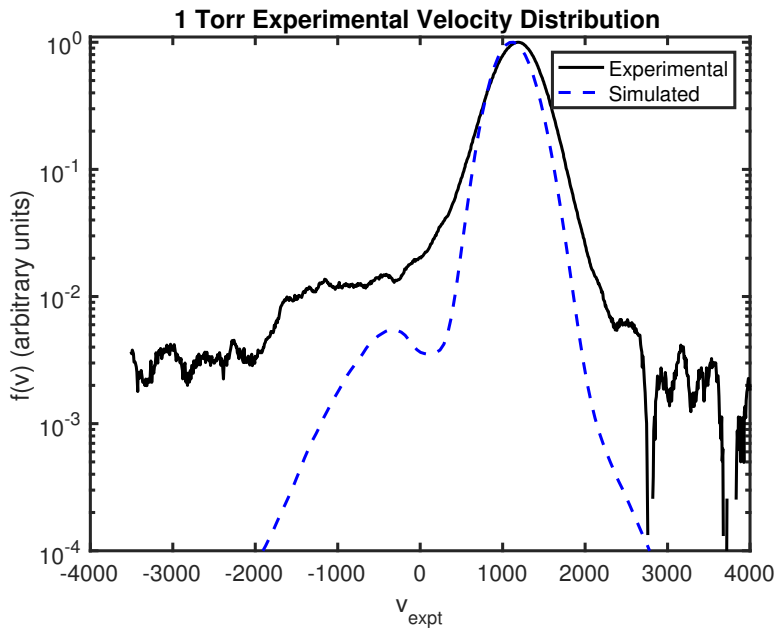

Figure 1.5 A plot of the velocity distribution at a point 3 mm upstream of the skimmer cone. One can easily see the difference between the simulated and experimental curves.

1.4 Previous work

Ross Spencer and Jaron Krogel wrote the majority of FENIX in 2005 [2]. Since its creation, FENIX has gone through some significant changes. To reduce computation time, Daniel Wilcox made it parallel using OpenMPI in 2009. The focus of this thesis is changes made to the subroutine handling the particle movement, whose code is included in Appendix A.

W. Neil Radicic measured the supersonic expansion in the area after the sampler cone [3]. Radicic did this by sweeping a range of frequencies with a laser pointed at the expansion region. Metastable argon fluoresces at 842.465 nm, so when the Doppler-shifted laser reached that wavelength in the frame of a moving atom, that atom fluoresced. This fluorescence was captured

and recorded for each laser frequency. This data can be seen in Figure 1.6. The graph[1] shows a combination of two Gaussian curves, showing two populations of atoms near the Mach disk, fast and slow. This means that the Mach disk is not a hard boundary, but rather a gradual mixing of the incoming beam and the background gas. One large difference between Radicic's work and this thesis is that Radicic did not include a skimmer cone in his measurements, as in the left half of Figure 1.2. An observant reader may notice that Figure 1.6 is flipped with respect to all other frequency spectrum graphs in this paper. This graph plots the frequency shift required by the laser to induce fluorescence, whereas the other plots show the Doppler shift from the particles.

Taylor and Farnsworth took data near the skimmer cone to determine speeds of metastable argon particles [4]. Their research is similar to that of [3], however, they included the effects of the skimmer on the shock structure. They used the same method of LIF, but with a different setup, as seen in Figure 1.7. They also compared a variety of different skimmer cone geometries. The results of the work for the skimmer used in this thesis are shown in Figure 1.8. Once again, the tail can be seen to the side of the main peak for each pressure.

[1] A note to the reader: Sometimes velocity and frequency shift are used interchangeably. Although the frequency shift is the quantity actually measured in the experiments, the velocity is found easily using $v = \lambda \cdot \Delta f \cos(\theta)$, where $\lambda = 842.465$ nm.

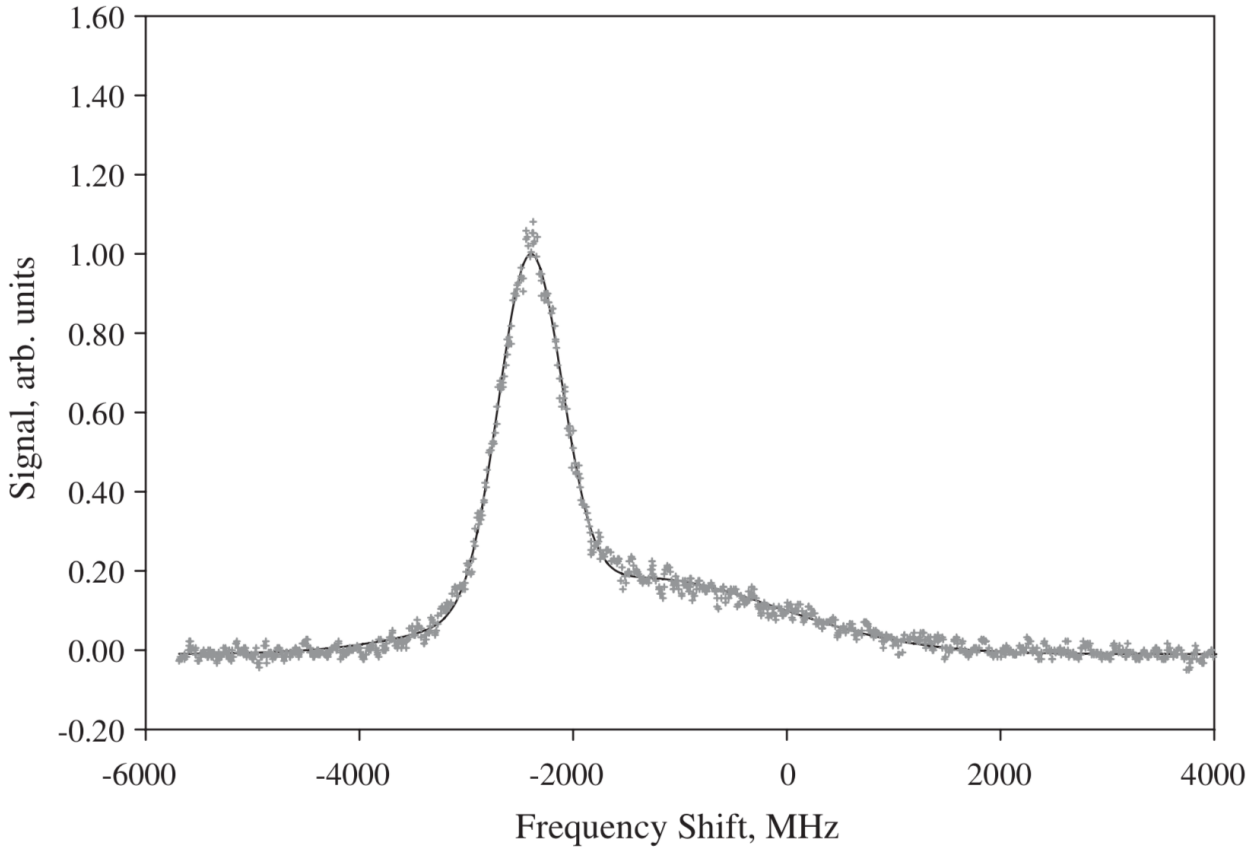

Figure 1.6 The Doppler-shifted fluorescence spectrum shows the frequency shift required to fluoresce argon metastable atoms. From the cited paper by Radicic, et. al.

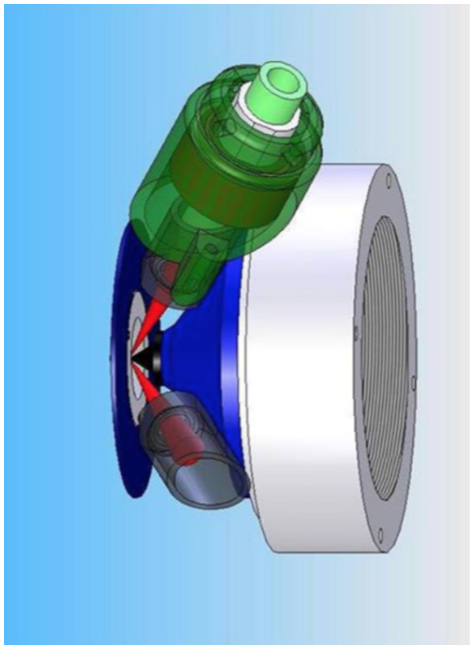

Figure 1.7 Mechanical drawing of the first stage of the vacuum interface, showing the skimmer conical support plate (blue), the support for the excitation optics (green), and the cones defining the excitation and emission beams (red). Used by Farnsworth and Taylor.

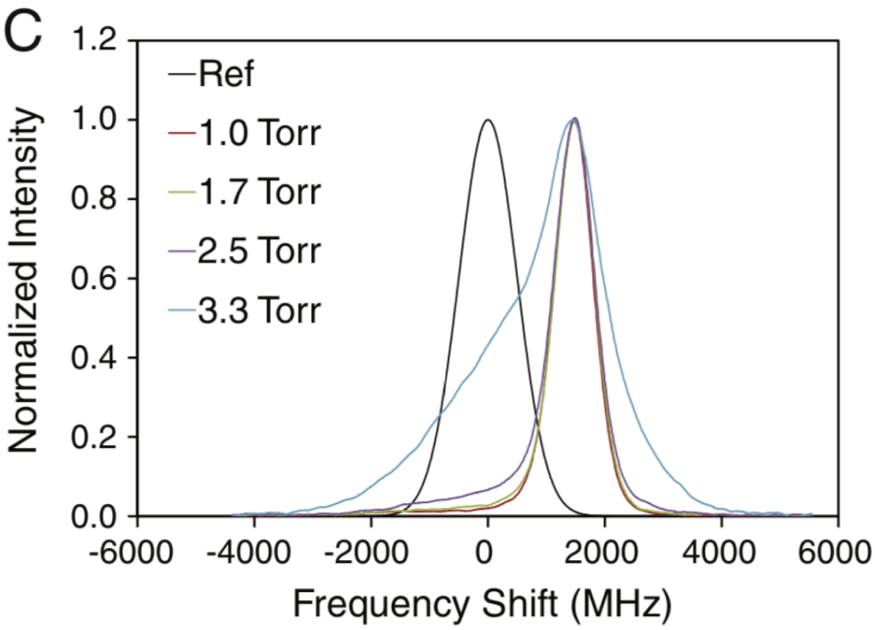

Figure 1.8 Velocity distributions of argon metastable atoms upstream of the skimmer cone used in this study. The pressures refer to the background pressure in initially in the vacuum interface. From the cited paper by Taylor, et. al.

1.5 Overview

To test the accuracy of different models, they are implemented in FENIX and the simulated results are compared against the experimental data taken by Taylor and Farnsworth. First, simulation parameters are adjusted to get the correct shape of the central peak, in section 2.1. Then, each of three models are compared in this thesis: Specular in section 2.3, Thermal in section 2.4 (also called Maxwell), and Cercignani-Lampis-Lord (CLL) in section 2.5. The relative accuracies and pitfalls of each model are then described in chapter 3. Finally, the future work needed to advance this research is discussed.

Chapter 2

Methods

In this chapter, I discuss the different methods I use in attempting to match the simulation to the experimental data. First, the instrumental broadening effects are explained. Then descriptions of the unknown physical constants are given, including the pressure inside the simulation, the temperature of the incoming gas. Lastly, I describe the three different gas-surface interaction models tested.

In Figure 1.5, the central peak does not match the experimental data. To match the central peak of the simulation to the data, the simulation must be made to match the conditions of the experiment. A simple summary of the steps taken to achieve this and their effects follows. First: Farnsworth's collection optics were off the axis by 63.5°, shifting the main peak closer to 0. Second: his optics were not infinitesimally narrow, giving rise to a broadened central peak. Third: the temperature of the incoming gas is not accurately known and changes the RMS velocity of the peak. Lastly: the background pressure changes the relative height of the tail, and is not well known in the experiment. Once these corrections have been implemented into the simulation, the central peaks of the simulation and experiment match closely, as can be seen in Figure 2.2.

2.1 Instrumental Broadening

As seen in Figure 1.8, the excitation and emission beams are both at an angle of 63.5° from the horizontal axis. In FENIX, the velocity of each particle is stored as a vector containing the three orthogonal elements in the lab frame: x, y and z. However, the measurements taken by Radicic [3] and Farnsworth [5] only result in a distribution of velocities towards or away from the measurement optics, requiting a change of axes to match the simulation and experimental data. This shifts the central peak closer to 0, because the measurement optics only detect the portion of the velocity pointed towards the optics. The majority of the gas is headed down the central axis, 63.5° away from the axis of the optics, therefore the velocities measured by the optics are shifted towards 0.

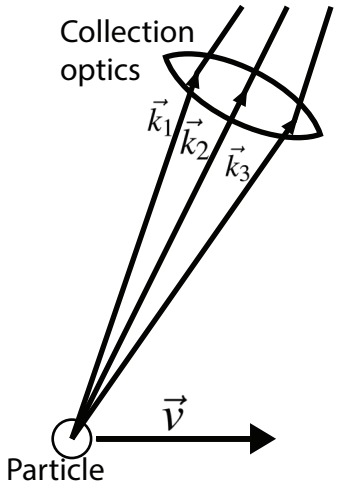

Figure 2.1 A ray diagram showing optical broadening in FENIX. Photons from the fluorescing argon metastable particles enter all across the collection optics. This distribution is simulated by creating multiple k vectors. The actual calculation of the measured velocity IN FENIX is done simply using $\vec{k}_i \cdot \vec{v}$.

The Doppler-shifted fluorescence frequency spectrum captured by Farnsworth is artificially broadened. The fluorescence radiates photons isotropically. An arbitrary number of photons entering the optics are selected from the isotropically radiated fluorescence. The photons are not actually simulated in FENIX, but the effects of their spreading across the optics are. A diagram of this

process can be seen in Figure 2.1 and is easily accomplished in FENIX by creating a series of k-vectors through the optics and taking their dot product with the particle's velocity. The effect of this change to simulate the finite size of the optics is to increase the width of the central peak beyond what would be observed due to temperature.

2.2 Unknown Physical Constants

The temperature upstream from the sampler cone was not measured in the experiment but can be found from the position of the actual peak. The simulation is adjusted to match this temperature. The RMS velocity of a gas in thermal equilibrium is given by $\sqrt{\frac{3k_BT}{m}}$, where k_b is Boltzmann's constant, T is the temperature of the gas, and m is the molecular mass. Thus, the temperature of the gas determines the RMS velocity. The main peak in Figure 1.5 consists of particles that have come out of the sampler cone and have not interacted with anything yet—neither the skimmer cone, nor other backwards-scattered particles–so their velocity is determined by the upstream temperature.

The pressure that should be used at the edge of the simulation is unknown because the experimental pressures are measured at a distant point. As is seen in Figure 1.1, the vacuum interface is connected by a long tube to the pump. FENIX simulates a region 10 mm from the radial axis. The experimental pressures are measured at a point near the pump itself. Due to the asymmetric nature of the vacuum connection, the actual pressure at the simulation's boundary cannot be accurately predicted. To determine the simulation pressure that should be used, I increase the pressure until reasonable results are reached. "Reasonable results" means that the average value of the tail on the left side of the graphs (such as Figure 1.5) is the same, even though the shape may still differ. We have found that the best background pressure for the specular and thermal models to be 2 Torr.

Correct simulation of these effects is crucial to matching the width of the simulated velocity distribution to the measured one. In Figure 2.2 one can see the clear differences that these effects

make on the simulated measurements. The center of the main peak was changed by implementing the off-axis optics and by varying the upstream temperature of the gas. The width of the peak is matched by expanding or contracting the finite size of the optics. And finally, the left side tail is raised to the correct average value by modifying the boundary pressures.

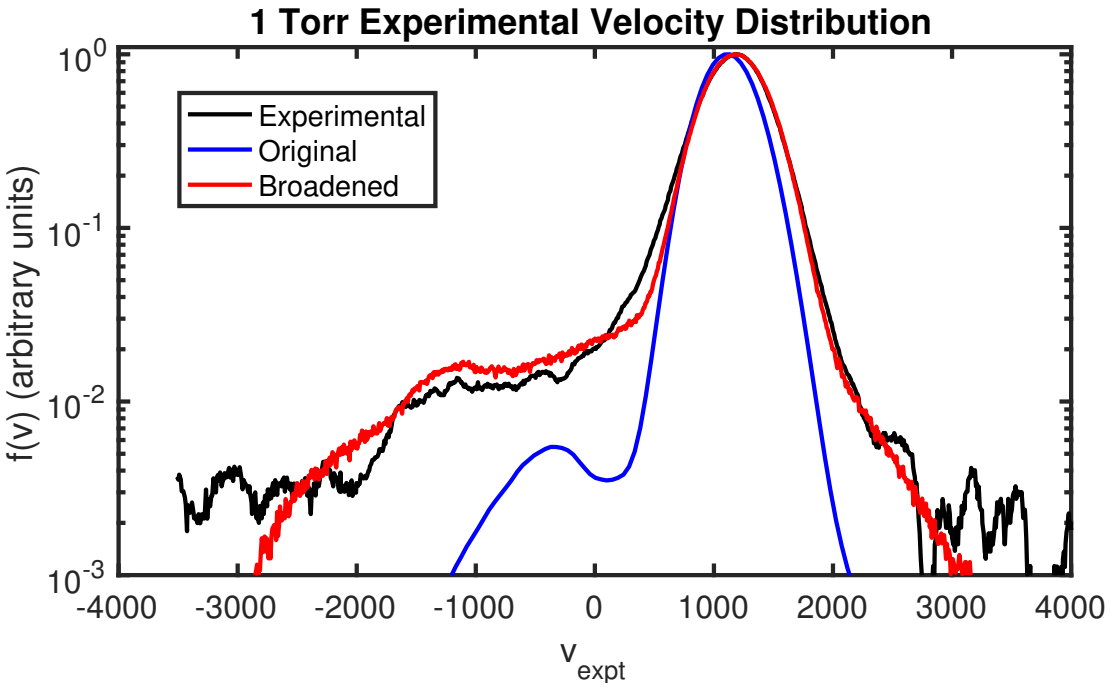

Figure 2.2 The effects of instrumental broadening and correctly simulated physical constants are seen compared with un-broadened simulations and the experimental data. The peak actually corresponds to a particle velocity of about 2,300 m/s but is shifted down due to the instrumental effects.

2.3 Specular Reflection Model

The specular model approximates the surface as perfectly flat and the collision as perfectly elastic. An illustration of the specular model is shown in Figure 2.3. A particle collides with the surface and bounces off with an angle equal to the incident angle and the same speed.

The simplest of the three methods, the specular model is implemented by switching the sign

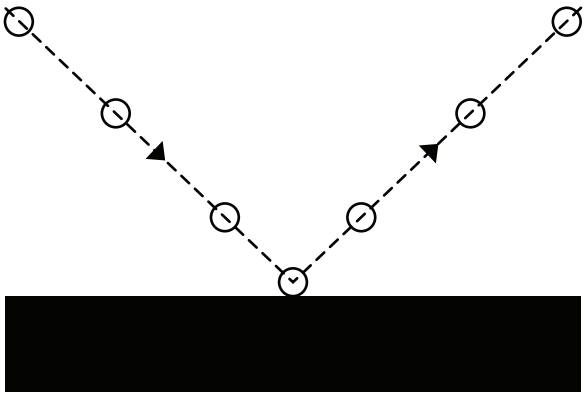

Figure 2.3 A simple figure capturing the essence of the specular model as a particle bounces off a surface in a perfectly elastic collision.

of the particle's velocity component normal to the surface. In FENIX, this is done by rotating the particle velocity into a frame normal to the incident surface. The normal velocity's sign is swapped and the velocities are then rotated back to the lab frame (x, y, z). This results in no energy lost by the particles, giving a larger population of high-energy backscattered particles.

2.4 Thermal Reflection Model

The thermal model adds in the facts that the surface is not flat and that the particle exchanges thermal energy with the surface. The basic concept is that a particle encounters a surface, is trapped somewhat in the microscopic imperfections, exchanges thermal energy, and is finally emitted some time later once it has come to thermal equilibrium with the surface. Since so many particles are interacting with the surface, the delay is removed in the simulation. This is the most widely-used method of modeling gas-surface interaction.

A compromise between simplicity and accuracy, the thermal model uses probabilistic methods to randomly choose a new speed and direction away from the surface. The new reflected angle is chosen with equal probability over a hemisphere. The two velocity components are determined by selecting a random point on a Maxwell-Boltzmann distribution. An example probability distribution

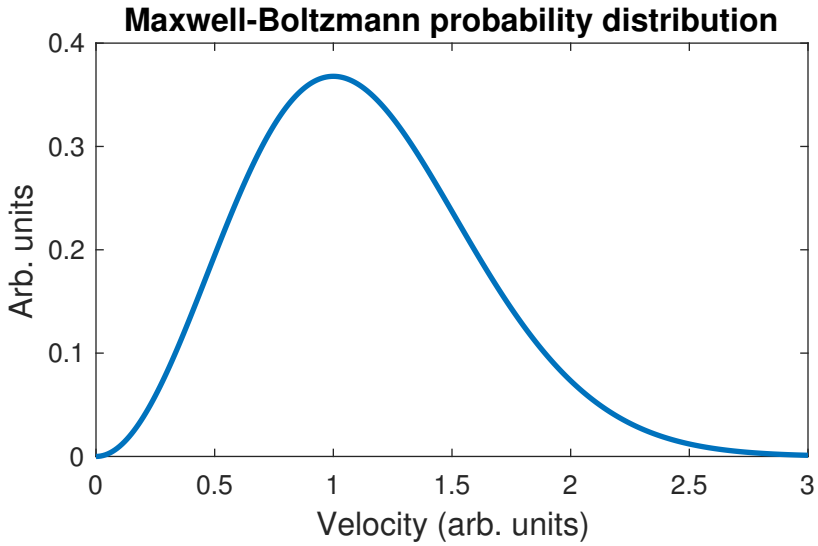

Figure 2.4 In the thermal model, the reflected particle velocities from the gas–surface collisions are determined using a Maxwell-Boltzman curve as is shown here.

can be seen in Figure 2.4. The Maxwell-Boltzmann probability density function is:

$$f(v) = \left(\frac{m}{2\pi k_B T}\right)^{3/2} 4\pi v^2 e^{-\frac{mv^2}{2k_B T}} \tag{2.1}$$

This gives the probability that a particle is at a given velocity for a certain mass and temperature. Therefore, the thermal model results in two combined Gaussian peaks: one composed of the particles in the gas beam from the sampler cone that have not interacted with anything, and the particles that have bounced off the skimmer cone surface.

2.5 CLL Reflection Model

The Cercignani-Lampis-Lord offers a smooth compromise between the thermal and specular models. As seen in Figure 2.5, the CLL model preferentially scatters particles forward. This means that each particle maintains a larger amount of energy, while still losing some to the surface through a thermal exchange. The scattering kernel for the CLL model is complicated; the interested reader is directed to the work of Sharipov [6] or Lord himself [7]. The implementation of the CLL model

into FENIX follows the method described by Padilla [8].

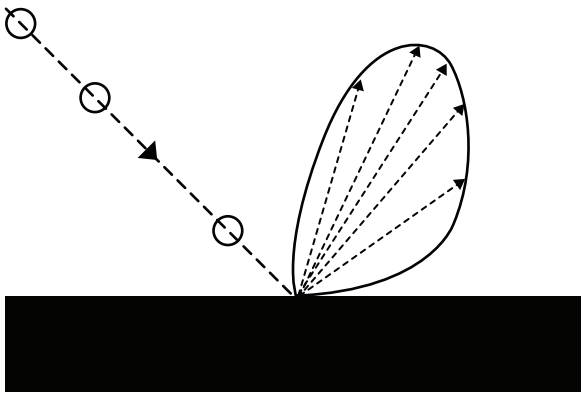

Figure 2.5 A picture showing roughly the probability distribution of the reflected particle as it leaves the surface. The CLL model's accommodation coefficients determine the exact shape of the scattering distribution.

Using two momentum accommodation coefficients—one in the tangential direction, and one in the normal direction—many effects of roughness and temperature on a surface can theoretically be modeled. The effects of the two coefficients is displayed in Figure 2.6. The tangential coefficient, σ_t, ranges from 0 to 1, corresponding to forward and diffuse scattering, respectively. The normal coefficient, σ_n ranges from 0 to 1, corresponding to a perfectly specular or a perfectly thermal reflection, respectively.

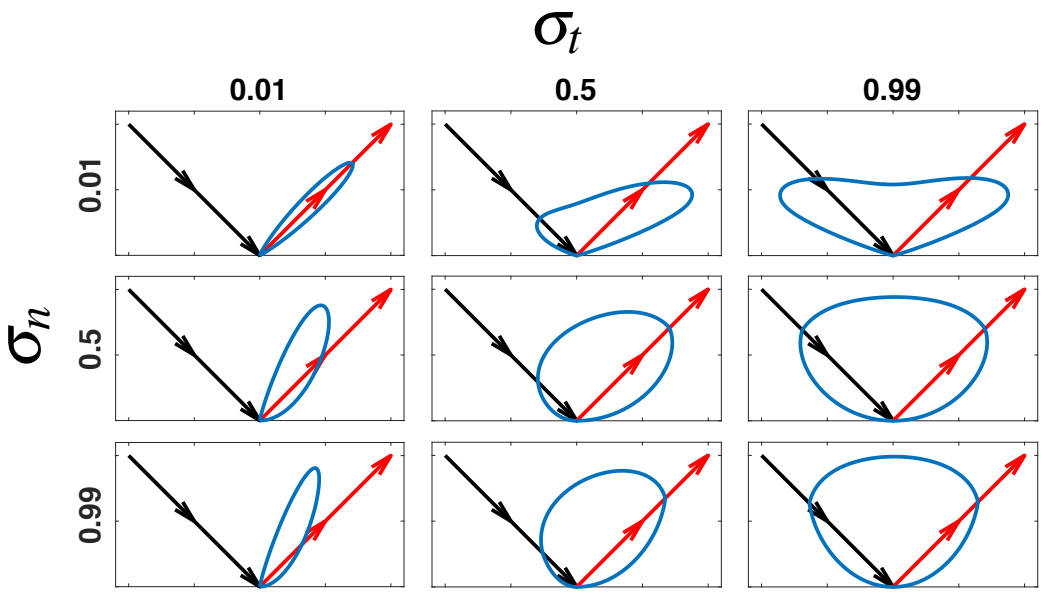

Figure 2.6 This series of pictures shows the influences of the two accommodation coefficients. σ_n is the normal accommodation coefficient and σ_t is the tangential one. The black arrow indicates the incident angle of the particle. The red arrow shows how the particle would bounce off in a perfectly specular reflection. The blue shape shows the probability distribution of the reflection angle in the CLL model.

Chapter 3

Results and Conclusions

This chapter contains a discussion of the results from implementing the aforementioned gas-surface interaction models in FENIX. First the specular model is discussed, followed by the thermal model and lastly the CLL model. A summary and comparison of these three models concludes this chapter.

3.1 Specular Model

The results from using the specular reflection model can be found in Figure 3.1. The first check that should be made with the results is the shape and position of the largest peak as compared to the same peak of the experimental data. The two overlay each other nicely, matching closely until the frequency drops to below 1 GHz. The specular model underestimates the number of particle in the region from about .25 GHz to 1 GHz. Below .25 GHz, the specular model overestimates the number of particles. This means that not enough energy is lost to the skimmer cone, since there are too many high-energy, backscattered particles. In addition, there aren't enough particle-particle collisions, because the scattered particles quickly leave the volume close to skimmer cone before they can interact with the main gas beam.

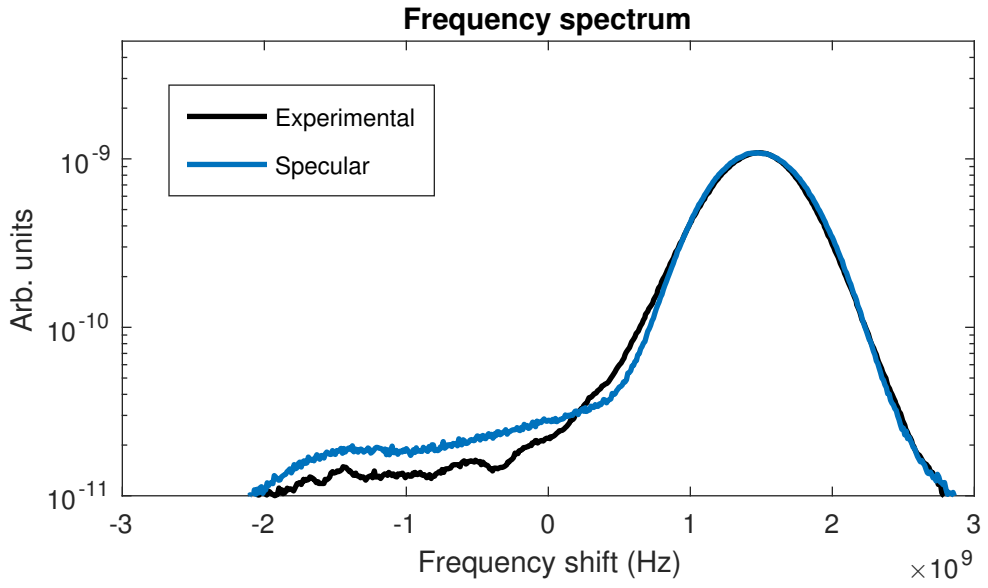

Figure 3.1 The specular reflection model leads to a much larger range of velocities away from the detector. This was simulated using a background pressure of 2 Torr.

3.2 Thermal Model

The results from using the thermal reflection model can be found in Figure 3.2. As before, the shape of the simulated central peak matches the experimental data closely. The general shape of the velocity distribution can be approximated by an addition of two Boltzmann distributions—one for the unperturbed gas beam coming from the sampler cone (on the right), and one for the particles reflected off the skimmer cone with lower energy. However, the tail does not match up quite as well. There aren't enough high-energy particles scattered off of the skimmer cone.

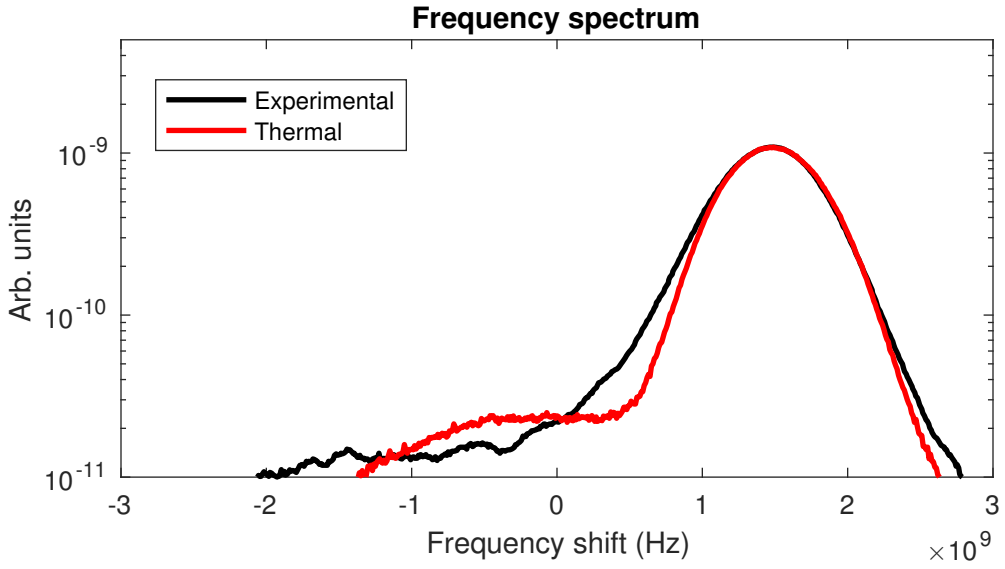

Figure 3.2 The thermal reflection model results show a superposition of two Maxwell-Boltzmann distributions. This was simulated using a background pressure of 2 Torr.

3.3 CLL Model

The results from using the CLL reflection model can be found in Figure 3.3. Again, the simulated central peak matches closely with the experimental peak. The tail does not match the experimental data, because the correct physical parameters (principally the pressure) have not been found. In addition, further testing is needed to ascertain the optimal accommodation coefficients, as well as the correct background pressure to use for the CLL interaction model.

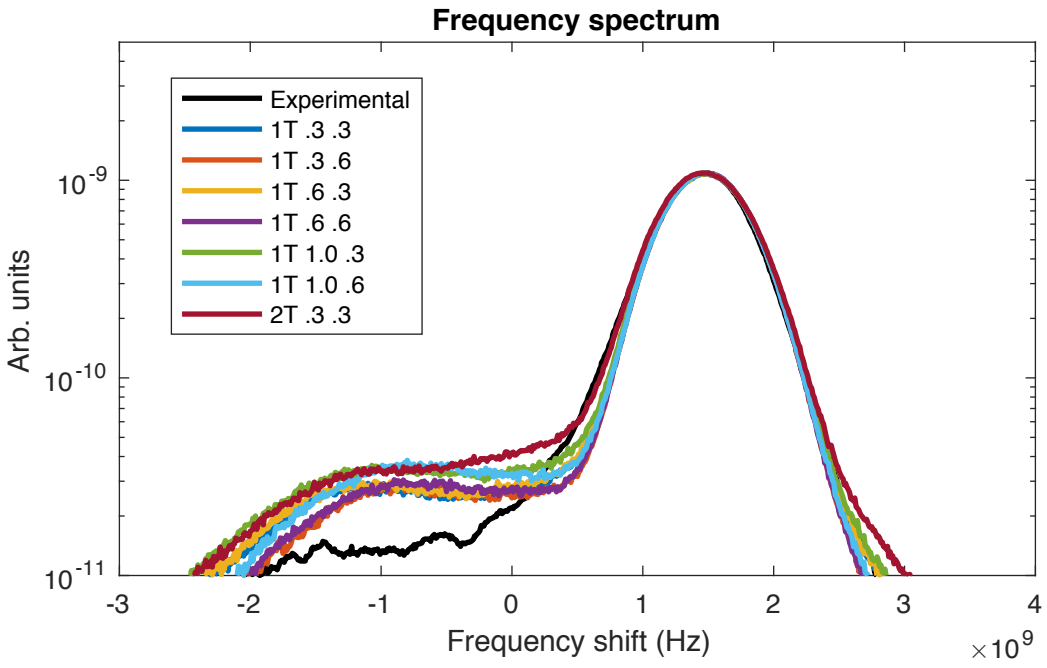

Figure 3.3 The left tail of the CLL model does not match the experimental data well. The legend entries read: Pressure (in Torr), σ_t, σ_n.

3.4 Summary

The superposition of the velocity distributions given by the three different models onto the experimental data is shown in Figure 3.4. A summary of the results from the three models follows.

Specular is the simplest model. It is incredibly easy to implement the specular model into a simulation, especially in the simple case of a sphere reflecting off a linear surface. However, it relies on some large simplifying assumptions: the surface is perfectly flat and only perfectly elastic collisions occur. These assumptions make it easier to code, and it yields the most accurate results.

Thermal is a middle ground in terms of assumptions, removing the assumptions about the nature of the reflections that the specular model makes. To compensate, less restrictive assumptions are made. First, the surface is extremely rough and can reflect particles in any direction. Second, particle velocities are randomly chosen from a 2-D Maxwell-Boltzmann distribution. While better

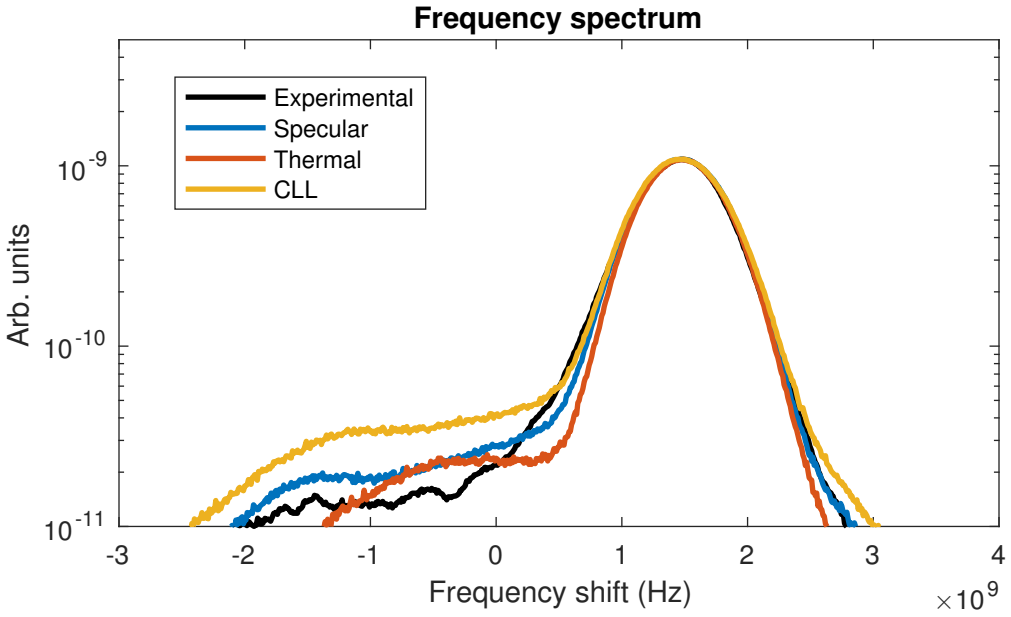

Figure 3.4 A comparison between the closest of the three models studied in this thesis and the experimental data found by Taylor and Farnsworth. All three simulated models were run with a background gas of 2 Torr. The CLL model used accommodation coefficients of $\sigma_t = 0.3$ and $\sigma_n = 0.3$.

than the specular model, the thermal model cannot account for the higher energy particles scattered off the surface as found in the experimental data.

CLL, while offering the most versatility, does not do the best job of reproducing the shock structure found in the experimental data. It is also the most difficult to implement, roughly three times as difficult as the specular model, based on the amount of code required for it to function.

3.5 Future Work

All three models overestimated the number of particles scattered back from the skimmer cone. In FENIX, we implemented the geometry of a new Finnigan standard cone, as seen in the left half of Figure 3.5. However, through the use of the ICP-MS, the skimmer cone is bombarded with 2,000 m/s gas at 700 K. This leads to the skimmer cone being slightly melted and the sharper features of

the skimmer cone tips are removed, looking more like the right half of Figure 3.5. Since the flat front of the skimmer cone is gone, the particles do not have that face to bounce off of. Less particles are scattered back towards the sampler cone, disturbing the shock structure less than if the flat front was still there.

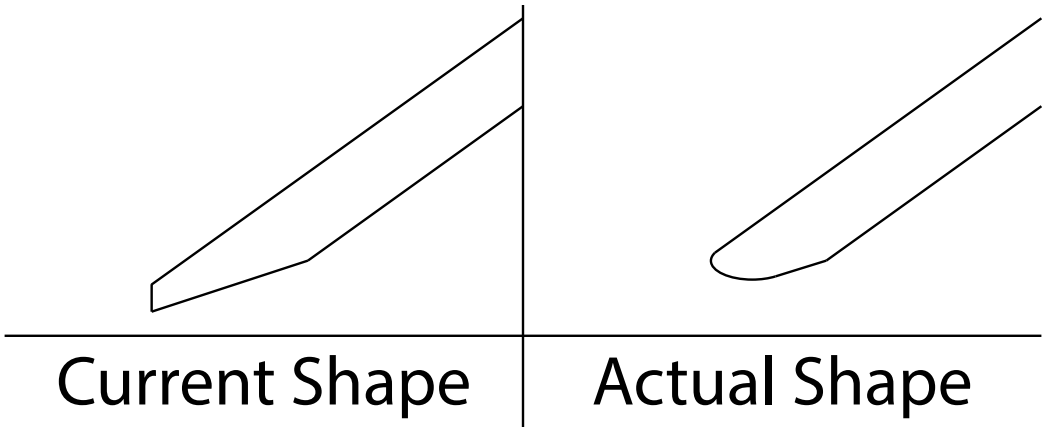

Figure 3.5 On the left, the geometry of a new Finnigan standard skimmer cone is shown. After being used, the front of the cone is melted, turning into something similar to the right side.

Currently, the flat face of the skimmer cone spans between two grid points, a distance of two microns. To implement a semi-melted geometry in FENIX, we would need approximately five times the grid point resolution. Some preliminary testing is shown in Figure 3.6. We have tested the flat front that has been used in the rest of this thesis, a rounded-tip cone, and a cone that comes to a sharp point. These were done at a background pressure of 2.5 Torr with the CLL model, which accounts for the larger tail, as compared to previous plots of frequency spectrums. There is clearly a large effect, that has gone unaccounted for in this research, which comes from the true geometry of the skimmer cone face.

The four steps to complete this project are as follows:

- Determine correct CLL parameters
- Resolve appropriate background pressure

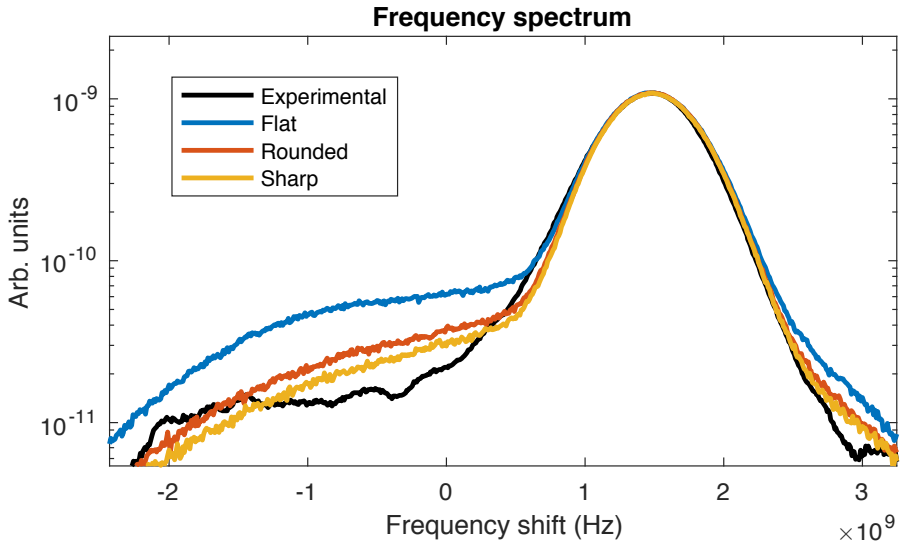

Figure 3.6 This graph shows the frequency distribution of the shock structure using different geometries of skimmer cone faces as compared with the experimental data.

- Accurately model the melted skimmer cone tip
- Complete enough runs to reach consistent results

Appendix A

Code

```
!----- move_mod -----
! This fortran module defines the routine for moving, including interactions with walls
module move_mod
    use helpers_mod
    implicit none
contains
    !----- move -----
    ! this subroutine makes all of the collision cells move their particles
    subroutine move(vxmin,dvx,vzmin,dvz,fdist,fradicic,Nx,Nz,Nn,Ndist,dvn,vnmin,noverlap)
        use core_data_mod
        use bounds_mod
        use simulation_mod
        use parallel_info_mod
        implicit none
        real(8) dN,dE,Nnet,Enet
        integer i, noverlap

        ! velocity distribution
        real(8) vxmin,dvx,vzmin,dvz,dvn,vnmin
        integer Nx,Nz,Nn
        integer Ndist
        real(8) fdist(Ndist,Nz,Nx),fradicic(Ndist,Nn)

        Nnet=0.d0
        Enet=0.d0
        dN=0.d0
        dE=0.d0

        noverlap=0  ! number of particles in Doppler optics overlap region, updated by cell_move
        do i=1,num_cells
            call cell_move(cells(i), tau,dN,dE,vxmin,dvx,vzmin,dvz,fdist,fradicic,Nx,Nz,Nn,Ndist,dvn,vnmin,noverlap)
            Nnet=Nnet+dN
            Enet=Enet+dE
        end do
        do i=1,num_gc
            call cell_move(gc(i), tau,dN,dE,vxmin,dvx,vzmin,dvz,fdist,fradicic,Nx,Nz,Nn,Ndist,dvn,vnmin,noverlap)
            Nnet=Nnet+dN
            Enet=Enet+dE
        end do
    end subroutine move
    !--- cell_move ---
```

```
45      ! Moves all the particles in the cell, performing interactions with the metals
        subroutine cell_move(cl, tau,dN,dE,vxmin,dvx,vzmin,dvz,fdist,fradicic,Nx,Nz,Nn
            ,Ndist,dvn,vnmin,noverlap)
            use core_types_mod
            use core_data_mod
            use helpers_mod
50          use constants_mod
            use bounds_mod
            use parallel_info_mod
            use simulation_mod
            implicit none
55
            !inputted values
            type(cell_type) cl

            type(particle_type) p0
60          real(8) tau
            integer Ndist

            ! we will know the details of a mask like this: masks(cl%mask)
            type(mask_type) msk
65          type(particle_type) temp
            real(8) th1, th2, u1, u2
            real(8) answer,dr,dz,taubar,dperp1,dperp2,dperp,vtot2
            real(8) a,b,c,e,f,d,t1,t2,myhit,chk1,chk2,uh1,uh2
            real(8) thit,uhit,zhit,rhit,vx,vy,vz,di,x,y,vxp,vyp
70
            real(8) grhit,gzhit,wrhit,wzhit,vn,offset
            real(8) E1,E2
            integer i,j,k,l,count,klook,flag
            real(8) tmprand, tmprand2
75          real(8) vthwall, vg, vw, v_2D, theta
            real(8) Nin,Nout,Ein,Eout,zold,znew,zchkleft,dE,dN
            real(8) rnd,vth,phi,r0,z0,n0r,n0z

            integer numtrace,numargon,ichk
80
            ! velocity distribution
            real(8) vxmin,dvx,vzmin,dvz,zk(Ndist)
            integer Nx,Nz,Nn,ix,jn,kk,jz
            real(8) fdist(Ndist,Nz,Nx),fx,fz,fn,dzlook,fradicic(Ndist,Nn)
85          real(8) vsight,isighti
            real(8) thetarad, phirad,vzp,dvn,vnmin
            real(8) nphi, dphi,nc,dc
            real(8) vcs,vsn,csthetamax,thetamax,anorm
            real(8) cstheta,sntheta,vj
90          real(8) dthrot, thetar, theta0, phi0
            integer nrotate, nd, nv, dopplerskip, noverlap

            !probability distribution for different types of collisions
            real(8) therm, spec, ang, total
95          integer ispec,itherm,icll
            real(8) an,at,r1,t3,r3,t4,r5,t6,vtmp
            real(8) vmpw, vtan, vpar, vperp, cs, sn, rbwaist, dchk, xp, yp, zp, csp,
                snp
            real(8) beamangle, csbeam, snbeam, ktot, csphi, snphi, kxp, kyp, kzp, kx,
                ky, kz, domega, facs

100         ! load the zk array for watching the distribution function
            dzlook=1.0d-3
            do jz=1,Ndist
                zk(jz)=(jz-1.)*dzlook+7.8e-3
            end do
105
            ! variables for tracking particles and energy entering and leaving
            zchkleft = 1.038d-4*.5
            Nin=0.d0
            Nout=0.d0
110         Ein=0.d0
            Eout=0.d0
            ! set the size of the offset which will be used to move a particle that
```

```
          hits
        ! the wall a little bit into the cell before computing the next bounce
115     offset = (cl%max_z-cl%min_z)*1.d-4
        ! start of move if block
        ! if the mask code is 0, just move them
        !*********start mask=0, ballistic move and return ***************
120     if(cl%mask.le.0) then
            !****Start Ballistic move Loop over particles
            k = 1
            do while(k .le. cl%num_parts)
125             if( cl%ps(k)%element == 0 ) then
                    ! normal particle, ballistic move
                    cl%ps(k)%x = cl%ps(k)%x + cl%ps(k)%vx*tau
                    cl%ps(k)%y = cl%ps(k)%y + cl%ps(k)%vy*tau
130                 zold = cl%ps(k)%z
                    cl%ps(k)%z = cl%ps(k)%z + cl%ps(k)%vz*tau
                    znew=cl%ps(k)%z
                else
135                 ! trace particle, special move (1 of 2)
                    ! step 1: modify the velocity as a result of the electric
                       field
                    ! vnew_vector = vold_vector + (tr_q*tau/tr_m)*ElectricField_R(
                       position_old)
140                 cl%ps(k)%vx = cl%ps(k)%vx + (tr_q*tau/tr_m)*cl%efield_r
                    cl%ps(k)%vz = cl%ps(k)%vz + (tr_q*tau/tr_m)*cl%efield_z
                    zold = cl%ps(k)%z
                    ! step 2: Move the particle with the new velocity
145                 cl%ps(k)%x = cl%ps(k)%x + cl%ps(k)%vx*tau
                    cl%ps(k)%y = cl%ps(k)%y + cl%ps(k)%vy*tau
                    cl%ps(k)%z = cl%ps(k)%z + cl%ps(k)%vz*tau
                    znew = cl%ps(k)%z
                end if
150
                ! rotate the particle to theta=0 (x-z plane) where x=d and y=0
                ! also set ps%r=d
                d=sqrt(cl%ps(k)%x**2+cl%ps(k)%y**2)
155             di=1.d0/d
                vx=cl%ps(k)%vx
                vy=cl%ps(k)%vy
                cl%ps(k)%vx=di*(cl%ps(k)%x*vx+cl%ps(k)%y*vy)
160             cl%ps(k)%vy=di*(cl%ps(k)%x*vy-cl%ps(k)%y*vx)
                cl%ps(k)%x=d
                cl%ps(k)%y=0.d0
                cl%ps(k)%r=d
                !
                   **************************************************************************
                   e
165             ! special: start dump particles to look at the distribution
                   function
                !          fradicic contains velocity bins of width dvz running
                   from vxmin to (Nz-1)*dvz
                ! only do this check if the cell min_r is small enough for there
                   to be hope
                rbwaist = 0.5d-3
170             dchk = 1.5*rbwaist
                ! set dopplerskip to the number of time steps to skip between
                   Doppler output. Set it
                ! in the hundreds so that the observation region can be populated
                   with new particles
                ! set it ridiculously high if you don't want to do Doppler
175             dopplerskip = 1000
                if(cl%min_r.le.dchk.and.mod(cur_step,dopplerskip).eq.0) then  !
                   top of cell minimum radius check
                    ! also wait some steps before checking
```

```
                    ! f(v) so that we can get new particles
                    ! into the viewing window
180
                    do kk=1,Ndist  ! loop over viewing locations
                        ! check each of the Ndist windows along the axis and
                            collect f(vz,vr)
                        ! by casting each particle to the corners of the velocity
                            grid
185
                        ! We will rotate particles about the z-axis to simulate
                            the full collection
                        ! volume inside the intersecting beam waists of the optics
                            . For this selection
                        ! choose particles that are within 1.5 beam waist radii in
                            both r and z,
                        ! centered on zk(kk). Also restrict to argon, element 0
190
                        if(  cl%ps(k)%x.lt.dchk.and. abs(cl%ps(k)%z-zk(kk)).lt.
                            dchk  .and. cl%ps(k)%element.eq.0) then ! top: particle
                            selection if
                            noverlap=noverlap+1  ! watch now many particles are in
                                this collection volume
                            ! now generate rotated particles to fill in the
                                detection volume
195
                            nrotate = 3
                            dthrot = 2.d0*pi/real(nrotate)
                            call rand_mt(tmprand2)
                            theta0=dthrot*tmprand2
200
                            do nd=1,nrotate    ! top of particle rotation
                                thetar=nd*dthrot + theta0
                                csp=cos(thetar)
205                             snp=sin(thetar)
                                ! get the rotated particle positions, exploiting y
                                    =0
                                xp = cl%ps(k)%x*csp
                                yp = cl%ps(k)%x*snp
210                             zp=cl%ps(k)%z-zk(kk)
                                ! only keep this particle if it's in the
                                    intersecting cylinders of the two beam radii
                                beamangle = 63.5d0/180.d0*pi ! beam angle measured
                                    from the z-axis
                                csbeam = cos(beamangle)
215                             snbeam = sin(beamangle)
                                chk1 = (xp*csbeam-zp*snbeam)**2  +yp**2  ! x-z
                                    plane beam
                                chk2 = (yp*csbeam-zp*snbeam)**2  +xp**2  ! y-z
                                    plane beam
220                             if(chk1.le.rbwaist**2.and.chk2.le.rbwaist**2) then
                                    ! accept this particle into the detector

                                    ! now that we accept this particle, rotate the
                                        particle velocities too
                                    vxp=cl%ps(k)%vx*csp        +cl%ps(k)%vy*snp
225                                 vyp=-cl%ps(k)%vx*snp        +cl%ps(k)%vy*csp
                                    vzp= cl%ps(k)%vz
                                    ! now calculate this particle's contribution
                                        to the fluorescence signal by averaging
                                    ! over the angles of the excitation optics. Do
                                        this by finding the k-vector of
230                                 ! the exciting light so the the Doppler shift
                                        can simply be calculated via k(dot)v
                                    thetamax=atan2(6.35,63.5)
                                    thetamax = thetamax*1.4 ! extra broadening to
                                        match central width to expt, case C
```

```
                                csthetamax=cos(thetamax)
                                facs  = (1-csthetamax)
235
                                nphi=2
                                dphi=2*pi/nphi
                                nc=5
                                call rand_mt(tmprand2)
240                             phi0=dphi*tmprand2
                                dc=1.d0/nc
                                ktot = 2.*pi/(801.48d-9)  ! excitation
                                    wavelength, Radicic and Farnsworth
                                ! choose a set of k-vectors in a coordinate
                                    system aligned with the z-axis
245                             do j = 1,nphi !The azimuthal angle
                                    phirad = j*dphi + phi0
                                    csphi = cos(phirad)
                                    snphi = sin(phirad)
250                                 do i = 1,nc !The polar angle
                                        call rand_mt(tmprand2)
                                        cstheta = 1-tmprand2*facs
                                        sntheta = sqrt(1-cstheta**2)
255                                     kzp = cstheta*ktot
                                        kxp = sntheta*csphi*ktot
                                        kyp = sntheta*snphi*ktot
                                        ! rotate to the beam axis in the x-z
                                            plane
260                                     kx = csbeam*kxp - snbeam*kzp
                                        ky = kyp
                                        kz = snbeam*kxp+csbeam*kzp
                                        ! The Doppler shifted frequency is k(
                                            dot)v
265                                     domega = kx*vxp+ky*vyp+kz*vzp
                                    enddo ! bottom of theta loop, optics
                                enddo ! bottom of phi loop, optics
                            endif   ! bottom of rotated particle acceptance
                                into detection volume
                        enddo ! bottom of particle rotation
270                 endif ! bottom of particle selection if
                enddo ! bottom of Ndist loop over points along the axis to
                    check f(v)
            end if  ! bottom of cell minimum radius less than 1.5*beamwaist
            ! special: end dump particles at a spatial window to look at the
                distribution function
            !
                ****************************************************************
                e
275
            ! watch particles coming in from the left
            if( sign(1.d0,zchkleft-zold).ne.sign(1.d0,zchkleft-znew) ) then
                ! entering particle
280             if(cl%ps(k)%vz.gt.0.d0) then
                    Nin = Nin + 1.d0
                    Ein = Ein + .5d0*m*(cl%ps(k)%vx**2 + cl%ps(k)%vy**2 + cl%
                        ps(k)%vz**2 )
                else
                    Nout = Nout + 1.d0
285                 Eout = Eout + .5d0*m*(cl%ps(k)%vx**2 + cl%ps(k)%vy**2 + cl
                        %ps(k)%vz**2 )
                end if
            end if
            ! watch particles coming in from the left
290
            ! Checking if particles have moved out of the region,
            ! and if so, swapping them out
            ! This is because particles remaining in the cell are lined in the
                array from left
            !     to right, the max being "num_parts", while particles which
```

```
                !     need to be transferred to other cells (in module "
                   reload_cells")
                !     are lined in the array from right to left, the max being "
                   num_left"
                if(  cl%ps(k)%r .lt. cl%min_r .or. &
                cl%ps(k)%r .gt. cl%max_r .or. &
                cl%ps(k)%z .lt. cl%min_z .or. &
                cl%ps(k)%z .gt. cl%max_z ) &
                then
                !swap the current particle out
                if( cl%ps(k)%element == 0 ) then
                    ! normal particle
                    temp = cl%ps(k)
                    cl%ps(k) = cl%ps(cl%partition)
                    cl%ps(cl%partition) = cl%ps(cl%num_parts)
                    cl%ps(cl%capacity-cl%num_left) = temp
                    !update the counts
                    cl%partition = cl%partition - 1
                    cl%num_parts = cl%num_parts - 1
                    cl%num_left = cl%num_left + 1
                else
                    ! trace particle
                    temp = cl%ps(k)
                    cl%ps(k) = cl%ps(cl%num_parts)
                    cl%ps(cl%capacity-cl%num_left) = temp
                    !update the counts
                    cl%num_parts = cl%num_parts - 1
                    cl%num_left = cl%num_left + 1
                end if
            else
                k = k + 1 !Increment counter (instance 1 of 1 in this do loop)
            end if
        end do  !***end ballistic loop over particles
        return  ! all done if this is a ballistic cell
    end if ! bottom of mask<0 if block
    !***********end mask=0, ballistic ******************************

    ! if we get to here the mask code isn't zero
    ! unload the mask code
    msk=masks(cl%mask)
    ! this is a value which is needed to calculate thermal reflection.
    ! It is the thermal velocity of a particle at the temperature of a wall
    vthwall = sqrt( kb * msk%T / m )

    !********** start of loop over particles **********************
    k = 1
    do while(k .le. cl%num_parts)

        zold = cl%ps(k)%z

        !****FINDING OUT IF IT'S CLOSE ENOUGH TO HIT A WALL
        ! find the perpendicular distance to the boundary(ies)
        dperp1=(cl%ps(k)%r-msk%s0r)*msk%g1r + (cl%ps(k)%z-msk%s0z)*msk%g1z
        dperp1=abs(dperp1)

        if( abs(msk%w2r) .gt. 1.0d-9 .or. abs(msk%w2z) .gt. 1.0d-9 ) then
            dperp2=(cl%ps(k)%r-msk%s0r)*msk%g2r + (cl%ps(k)%z-msk%s0z)*msk%g2z
            dperp2=abs(dperp2)
        else
            dperp2=dperp1
        end if
        dperp=min(dperp1,dperp2)

        ! find the square of the particle speed
        vtot2=cl%ps(k)%vx**2+cl%ps(k)%vy**2+cl%ps(k)%vz**2

        ! if the particle can't go far enough in time tau to reach the wall
        ! along the shortest path it can't reach it at all - just move the
        ! particle
        if( vtot2*tau**2 < dperp**2 ) then
```

```
            ! move the particle ballistically
365         if( cl%ps(k)%element == 0 ) then
                ! normal particle, ballistic move
                cl%ps(k)%x = cl%ps(k)%x + cl%ps(k)%vx*tau
                cl%ps(k)%y = cl%ps(k)%y + cl%ps(k)%vy*tau
370             cl%ps(k)%z = cl%ps(k)%z + cl%ps(k)%vz*tau
                znew = cl%ps(k)%z
            else
                ! trace particle, special move (2 of 2)
                ! step 1: modify the velocity as a result of the electric field
375             ! vnew_vector = vold_vector + (tr_q*tau/tr_m)*ElectricField_R(
                    position_old)
                cl%ps(k)%vx = cl%ps(k)%vx + (tr_q*tau/tr_m)*cl%efield_r
                cl%ps(k)%vz = cl%ps(k)%vz + (tr_q*tau/tr_m)*cl%efield_z
                ! step 2: Move the particle with the new velocity
380             cl%ps(k)%x = cl%ps(k)%x + cl%ps(k)%vx*tau
                cl%ps(k)%y = cl%ps(k)%y + cl%ps(k)%vy*tau
                cl%ps(k)%z = cl%ps(k)%z + cl%ps(k)%vz*tau
                znew = cl%ps(k)%z
            end if
385
            ! rotate the particle to theta=0 (x-z plane) where x=d and y=0
            ! also set ps%r=d
            d=sqrt(cl%ps(k)%x**2+cl%ps(k)%y**2)
            di=1.d0/d
390
            vx=cl%ps(k)%vx
            vy=cl%ps(k)%vy
            cl%ps(k)%vx=di*(cl%ps(k)%x*vx+cl%ps(k)%y*vy)
            cl%ps(k)%vy=di*(cl%ps(k)%x*vy-cl%ps(k)%y*vx)
395         cl%ps(k)%x=d
            cl%ps(k)%y=0.d0
            cl%ps(k)%r=d
            ! this particle can't hit the wall, so we just moved it ballistically
400         ! Checking if particle has moved out of the region,
            ! and if so, swapping it out
            ! This is because particles remaining in the cell are lined in the
                array from left
            !     to right, the max being "num_parts", while particles which
            !     need to be transferred to other cells (in module "reload_cells")
405         !     are lined in the array from right to left, the max being "
                num_left"
            if(  cl%ps(k)%r .lt. cl%min_r .or. &
            cl%ps(k)%r .gt. cl%max_r .or. &
            cl%ps(k)%z .lt. cl%min_z .or. &
            cl%ps(k)%z .gt. cl%max_z ) &
410         then
            !swap the current particle out
            if( cl%ps(k)%element == 0 ) then
                ! normal particle
                temp = cl%ps(k)
415             cl%ps(k) = cl%ps(cl%partition)
                cl%ps(cl%partition) = cl%ps(cl%num_parts)
                cl%ps(cl%capacity-cl%num_left) = temp
                !update the counts
                cl%partition = cl%partition - 1
420             cl%num_parts = cl%num_parts - 1
                cl%num_left = cl%num_left + 1
            else
                ! trace particle
                temp = cl%ps(k)
425             cl%ps(k) = cl%ps(cl%num_parts)
                cl%ps(cl%capacity-cl%num_left) = temp
                !update the counts
                cl%num_parts = cl%num_parts - 1
```

```
                cl%num_left = cl%num_left + 1
            end if
        else
            k = k + 1 !Increment counter (instance 1 of 2 in this do loop)
        end if
        cycle
    end if
    !******** top of multiple bounce while loop*************
    ! set the total time of this step (handles multiple bounces)
    taubar=tau
    count=0
    do while (taubar.gt.0.d0)
        count=count+1
        if(count.ge.200) stop 678

        ! do the hard work of seeing if we hit the wall
        ! ********** time to hit segment 1 ************************************
        if(abs(msk%w1z) .le. 1.0d-9 .and. abs(msk%w1r) .le. 1.0d-9 ) then
            write(*,*) "mask doesn't exist! w1r and w1z both zero"
            stop 277

            ! vertical line is special
        else if(abs(msk%w1z) .le. 1.0d-9) then
            t1 = (msk%s0z-cl%ps(k)%z)/cl%ps(k)%vz  ! much less round-off error
            u1 = (sqrt((cl%ps(k)%x+t1*cl%ps(k)%vx)**2+ &
            (cl%ps(k)%y+t1*cl%ps(k)%vy)**2) - msk%s0r)/msk%w1r
            t2 = 1.d99 !just mark tau2 and s2 as infeasible
            u2 = -1

            ! check to make sure that u is positive and that the crossing time is
                positive
            if(t1.lt.0.d0.or.u1.lt.0.d0) t1=1.d99
            th1=t1
            uh1=u1
            ! general case: quadratic
        else
            a = msk%w1z**2*(cl%ps(k)%vx**2+cl%ps(k)%vy**2) - &
            (msk%w1r*cl%ps(k)%vz)**2
            e = msk%w1z*msk%s0r-msk%w1r*(msk%s0z-cl%ps(k)%z)
            f = msk%w1z*cl%ps(k)%x
            b=2*(msk%w1z*cl%ps(k)%vx*f-msk%w1r*cl%ps(k)%vz*e)
            c=(f+e)*(f-e)
            d=b**2-4*a*c
            if(d.lt.0.d0) then
                th1=1.d99
            else
                d=sqrt(d)
                a=0.5d0/a
                t1=(-b+d)*a
                u1 = (cl%ps(k)%z+cl%ps(k)%vz*t1-msk%s0z)/msk%w1z
                chk1 = msk%s0r+msk%w1r*u1
                t2=(-b-d)*a
                u2 = (cl%ps(k)%z+cl%ps(k)%vz*t2-msk%s0z)/msk%w1z
                chk2 = msk%s0r+msk%w1r*u2

                if(t1.le.0.d0.or.chk1.lt.0.d0.or.u1.lt.0.d0) t1=1.d99
                if(t2.le.0.d0.or.chk2.lt.0.d0.or.u2.lt.0.d0) t2=1.d99
                if(t1.lt.t2) then
                    th1=t1
                    uh1=u1
                else
                    th1=t2
                    uh1=u2
                end if
            end if
        end if
        ! set thit and the point on the segment where the crossing occurs
        ! if there is a valid crossing
        ! also load grhit and gzhit with the components of g, the inward pointing
```

```
! unit vector, at the contact point
if(th1.lt.1.d99.and.th1.lt.taubar) then
    thit=th1
    uhit=uh1
    zhit=msk%s0z+uhit*msk%w1z
    rhit=abs(msk%s0r+uhit*msk%w1r)
    grhit=msk%g1r
    gzhit=msk%g1z
    wrhit=msk%w1r
    wzhit=msk%w1z
else
    thit=1.d99
    zhit=0.d0
    rhit=0.d0
    grhit=0.d0
    gzhit=0.d0
    wrhit=0.d0
    wzhit=0.d0
end if
! ********** end   time to hit segment 1
    ************************************
! ********** begin time to hit segment 2
    ************************************
! segment 2 exists if the w2 unit vector is nonzero
if(abs(msk%w2r)+abs(msk%w2z).ge.1.0d-9) then
    ! vertical line is special
    if(abs(msk%w2z) .le. 1.0d-9)then
        t1 = (msk%s0z-cl%ps(k)%z)/cl%ps(k)%vz  ! much less round-off error
        u1 = (sqrt((cl%ps(k)%x+t1*cl%ps(k)%vx)**2+ &
        (cl%ps(k)%y+t1*cl%ps(k)%vy)**2) - msk%s0r)/msk%w2r
        t2 = 1.d99 !just mark tau2 and s2 as infeasible
        u2 = -1
        ! check to make sure that u is positive
        if(t1.lt.0.d0.or.u1.lt.0.d0) t1=1.d99
        th2=t1
        uh2=u1
        ! general case: quadratic
    else
        a = msk%w2z**2*(cl%ps(k)%vx**2+cl%ps(k)%vy**2) - &
        (msk%w2r*cl%ps(k)%vz)**2
        e = msk%w2z*msk%s0r-msk%w2r*(msk%s0z-cl%ps(k)%z)
        f = msk%w2z*cl%ps(k)%x
        b=2*(msk%w2z*cl%ps(k)%vx*f-msk%w2r*cl%ps(k)%vz*e)
        c=(f+e)*(f-e)
        d=b**2-4*a*c
        if(d.lt.0.d0) then
            th2=1.d99
        else
            d=sqrt(d)
            a=0.5d0/a
            t1=(-b+d)*a
            u1 = (cl%ps(k)%z+cl%ps(k)%vz*t1-msk%s0z)/msk%w2z
            chk1 = msk%s0r+msk%w2r*u1
            t2=(-b-d)*a
            u2 = (cl%ps(k)%z+cl%ps(k)%vz*t2-msk%s0z)/msk%w2z
            chk2 = msk%s0r+msk%w2r*u2
            if(t1.le.0.d0.or.chk1.lt.0.d0.or.u1.lt.0.d0) t1=1.d99
            if(t2.le.0.d0.or.chk2.lt.0.d0.or.u2.lt.0.d0) t2=1.d99
            if(t1.lt.t2) then
                th2=t1
                uh2=u1
            else
                th2=t2
                uh2=u2
            end if
        end if
    end if
```

```
            ! if segment 2 exists we need to choose the soonest crossing between
                segment 1 and 2
            ! set thit and the point on the segment where the crossing occurs
            ! if there is a valid crossing and if it happens sooner than that
            ! of segment 1
            ! also find the inward pointing g vector at the contact point
            if(th2.lt.th1.and.th2.lt.taubar) then
                thit=th2
                uhit=uh2
                zhit=msk%s0z+uhit*msk%w2z
                rhit=abs(msk%s0r+uhit*msk%w2r)
                grhit=msk%g2r
                gzhit=msk%g2z
                wrhit=msk%w2r
                wzhit=msk%w2z
            end if
        end if
        ! ********** end time to hit segment 2
            ************************************
        !*********** move the particles
        ! if thit=1.d99 use taubar up with a ballistic move
        if(thit.gt.1.d98) then

            cl%ps(k)%x = cl%ps(k)%x + cl%ps(k)%vx*taubar
            cl%ps(k)%y = cl%ps(k)%y + cl%ps(k)%vy*taubar
            cl%ps(k)%z = cl%ps(k)%z + cl%ps(k)%vz*taubar
            znew = cl%ps(k)%z

            ! rotate the particle to theta=0 (x-z plane)
            d=sqrt(cl%ps(k)%x**2+cl%ps(k)%y**2)
            di=1.d0/d

            vx=cl%ps(k)%vx
            vy=cl%ps(k)%vy
            cl%ps(k)%vx=di*(cl%ps(k)%x*vx+cl%ps(k)%y*vy)
            cl%ps(k)%vy=di*(cl%ps(k)%x*vy-cl%ps(k)%y*vx)
            cl%ps(k)%x=d
            cl%ps(k)%y=0.d0
            cl%ps(k)%r=d
            ! set taubar to zero because it's used up
            taubar=0.d0
        else
            count = count + 1  ! counting how many bounces the particle had
            !********begin SPECULAR/THERMALIZED REFLECTION***********
            ! We already know the location of the collision with the wall (rhit,
                zhit, from above)
            ! Now we just need to calculate the reflection off the wall
            !These variables control the probability distribution for the
            !collision types
            icll=1  ! use CLL
            itherm=0 ! use thermal
            ispec=0 ! use specular

            if( icll.eq.1       ) then
                !this is the CLL model
                ! put the particle at the segment contact point, reset its
                    velocity,
                ! and decrement taubar
                ! move the particle to the metal surface
                x = cl%ps(k)%x + cl%ps(k)%vx*thit
                y = cl%ps(k)%y + cl%ps(k)%vy*thit

                ! unload vx,vy,vz into more convenient variables
                vx = cl%ps(k)%vx
                vy = cl%ps(k)%vy
                vz = cl%ps(k)%vz

                ! find the normal (vg) and tangential (vw) components; note that
                    vy is also tangential
                ! also note that the normal unit vector g points into the metal
```

```
                vg = grhit*vx+gzhit*vz
                vw = wrhit*vx+wzhit*vz

                !These are the accommodation coefficients for CLL (an is normal,
                !at is tangential)
                at = .3d0
                an = .6d0

                !This is the most probable velocity for a 3D maxwellian
                !probability distribution (follow _Padilla and Boyd)
                vmpw = sqrt( 2.d0 * kb * msk%T / m )

                ! Calculates the normal velocity
                call rand_mt(tmprand2)
                r1 = sqrt(-an*log(tmprand2))
                call rand_mt(tmprand2)
                t2 = 2.d0*pi*tmprand2
                vtmp = abs(vg/vmpw)*sqrt(1-an)
                vg = vmpw*sqrt(r1**2+vtmp**2+2.d0*r1*vtmp*cos(t2)) ! reassign the
                    inward normal velocity

                !This is the total tangential velocity and the cosine and sine of
                    the angle
                ! between the incoming velocity direction and w (in the x-z plane)
                vtan = sqrt(vw**2 + vy**2)
                cs = vw/vtan ! cosine
                sn = vy/vtan ! sine

                ! Calculates the parallel (to the incoming velocity) tangential
                    velocity
                call rand_mt(tmprand2)
                r3 = sqrt(-at*log(tmprand2))
                call rand_mt(tmprand2)
                t4 = 2.d0*pi*tmprand2
                vtmp = abs(vtan/vmpw)*sqrt(1.d0-at)
                vpar = vmpw*(vtmp+r3*cos(t4))

                ! Calculates the tangential velocity perpendicular to the incoming
                     velocity
                call rand_mt(tmprand2)
                r5 = sqrt(-at*log(tmprand2))
                call rand_mt(tmprand2)
                t6=2.d0*pi*tmprand2
                vperp = vmpw*r5*cos(t6)

                !Rotate the two tangential velocities, so that they
                !correspond with w and y in the simulation
                cl%ps(k)%vy = vpar*sn + vperp*cs
                vw = vpar*cs - vperp*sn

                ! now rotate (vw,vg,vy) back into the (x,z,y) coordinate system (y
                     is easy)
                cl%ps(k)%vx=-(vg*wzhit+vw*gzhit)/(grhit*wzhit-wrhit*gzhit)
                cl%ps(k)%vz=(vg*wrhit+vw*grhit)/(grhit*wzhit-wrhit*gzhit)

            else if(itherm.eq.1         ) then

                !thermalized reflection
                ! find a random angle, theta
                call rand_mt(tmprand2)
                theta = 2*pi*tmprand2

                ! sample a 2D maxwellian speed distribution
                call rand_mt(tmprand2)
                v_2D = vthwall*sqrt(-2.d0 * log(tmprand2))

                !since vy is always parallel to the wall, we can set vy right now
                cl%ps(k)%vy = v_2D*cos(theta)

                ! vw is referring to the other parallel direction, parallel to the
                     surface, and perpendicular to vy
                ! vw is a scalar
                vw = v_2D * sin(theta)
```

```
710
                ! vg is referring the normal distribution, which is a 2D
                    maxwellian speed distribution (not obvious)
                ! vg is a scalar
                call rand_mt(tmprand2)
                vg = vthwall*sqrt(-2.d0 * log(tmprand2)) ! again but with a new
                    random number
715             cl%ps(k)%vz = -vg*gzhit + vw*wzhit
                cl%ps(k)%vx = -vg*grhit + vw*wrhit
            elseif (ispec.eq.1) then
                ! put the particle at the segment contact point, reset its
                    velocity,
720             ! and decrement taubar
                ! move the particle in the x-y plane for rotation purposes
                x = cl%ps(k)%x + cl%ps(k)%vx*thit
                y = cl%ps(k)%y + cl%ps(k)%vy*thit
725             ! now rotate the velocities
                vxp = cl%ps(k)%vx
                vyp = cl%ps(k)%vy
                vz = cl%ps(k)%vz
730             d = sqrt(x**2+y**2)
                di = 1.d0/d
                vx = di*(x*vxp+y*vyp)
                vy = di*(x*vyp-y*vxp)
735
                ! specular reflection
                ! do the specular reflection using the vector formula, but make it
                    work
                ! for particles coming from inside the metal as well - for these
                    particles
                ! kill the normal component and then sent the particle off with |
                    vn|
740             ! away from the wall
                ! v(new) = v(old) - (vnormal+abs(vnormal))*g(unit normal into wall
                    )
                vn = grhit*vx+gzhit*vz
                vtmp=vn+abs(vn)
                cl%ps(k)%vx = vx-vtmp*grhit
745             cl%ps(k)%vz = vz-vtmp*gzhit
                cl%ps(k)%vy = vy
            else
                write(*,*) 'No wall interaction model chosen - time to die'
                stop 678
750         end if
            ! ********** end Specular/Thermalized reflection
            ! put the particle on the wall, and then just a little bit away
            ! in the direction of -g, so that thit won't come back zero next time
            cl%ps(k)%x = abs(rhit-grhit*offset)
755         cl%ps(k)%y = 0.d0
            cl%ps(k)%z = zhit-gzhit*offset
            znew = cl%ps(k)%z
            cl%ps(k)%r = cl%ps(k)%x
760         ! decrement taubar
            taubar=taubar-thit
        end if
    end do
    !****** bottom of multiple bounce while loop *********
765 ! watch particles coming in from the left
    if( sign(1.d0,zchkleft-zold).ne.sign(1.d0,zchkleft-znew) ) then
        ! entering particle
        if(cl%ps(k)%vz.gt.0.d0) then
770         Nin = Nin + 1.d0
            Ein = Ein + .5d0*m*( cl%ps(k)%vx**2 + cl%ps(k)%vy**2 + cl%ps(k)%vz**2
                )
        else
```

```
                Nout = Nout + 1.d0
                Eout = Eout + .5d0*m*( cl%ps(k)%vx**2 + cl%ps(k)%vy**2 + cl%ps(k)%vz
                    **2 )
775         end if
        end if
        ! watch particles coming in from the left
        !******** Checking for particles exiting the region,
        ! and if so, swapping them out
780     ! This is because particles remaining in the cell are lined in the array from
            left
        !     to right, the max being "num_parts", while particles which
        !     need to be transferred to other cells (in module "reload_cells")
        !     are lined in the array from right to left, the max being "num_left"
        if(  cl%ps(k)%r .lt. cl%min_r .or. &
785     cl%ps(k)%r .gt. cl%max_r .or. &
        cl%ps(k)%z .lt. cl%min_z .or. &
        cl%ps(k)%z .gt. cl%max_z ) &
        then
        !swap the current particle out
790     if( cl%ps(k)%element == 0 ) then
            ! normal particle
            temp = cl%ps(k)
            cl%ps(k) = cl%ps(cl%partition)
            cl%ps(cl%partition) = cl%ps(cl%num_parts)
795         cl%ps(cl%capacity-cl%num_left) = temp
            !update the counts
            cl%partition = cl%partition - 1
            cl%num_parts = cl%num_parts - 1
            cl%num_left = cl%num_left + 1
800     else
            ! trace particle
            temp = cl%ps(k)
            cl%ps(k) = cl%ps(cl%num_parts)
            cl%ps(cl%capacity-cl%num_left) = temp
805         !update the counts
            cl%num_parts = cl%num_parts - 1
            cl%num_left = cl%num_left + 1
        end if
    else
810     k = k + 1 !Increment counter (instance 2 of 2 in this do loop)
    end if
    end do
    !****** end of loop over particles **************
    dN=(Nin-Nout)*Nef
815 dE=Ein-Eout
    end subroutine cell_move
    end module move_mod
```

Bibliography

[1] G. A. Bird, *The DSMC Method* (CreateSpace Independent Publishing Platform, 2013).

[2] R. L. Spencer, J. Krogel, J. Palmer, A. Payne, A. Sampson, W. Somers, and C. N. Woods, "Modeling the gas flow upstream and in the sampling nozzle of the inductively coupled plasma mass spectrometer via the Direct Simulation Monte Carlo algorithm," Spectrochimica Acta Part B **64,** 215–221 (2009).

[3] W. N. Radicic, J. B. Olsen, R. V. Nielson, J. H. Macedone, and P. B. Farnsworth, "Characterization of the supersonic expansion in the vacuum interface of an inductively coupled plasma mass spectrometer by high-resolution diode laser spectroscopy," Spectrochimica Acta Part B **61,** 686–695 (2006).

[4] P. B. Farnsworth and R. L. Spencer, "Ion sampling and transport in ICP-MS," Spectrochimica Acta Part B **134,** 105–122 (2017).

[5] N. Taylor and P. B. Farnsworth, "Experimental characterization of the effect of skimmer cone design on shock formation and ion transmission efficiency in the vacuum interface of an inductively coupled plasma mass spectrometer," Spectrochimica Acta Part B **69,** 2–8 (2012).

[6] F. Sharipov, "Application of the Cercignani–Lampis scattering kernel to calculations of rarefied gas flows. II. Slip and jump coefficients," European Journal of Mechanics B/Fluids **22,** 133–143 (2003).

[7] R. G. Lord, "Some extensions to the Cercignani-Lampis gas-surface scattering kernel," Physics of Fluids A: Fluid Dynamics 3 (1991).

[8] J. F. Padilla and I. D. Boyd, *39th AIAA Thermophysics Conference* (2007).

Index