A HOMEGROWN DSMC-PIC MODEL FOR ELECTRIC PROPULSION PLUMES

A Thesis

presented to

the Faculty of California Polytechnic State University,

San Luis Obispo

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in Aerospace Engineering

by

Dominic Lunde

June 2019

COMMITTEE MEMBERSHIP

TITLE: A Homegrown DSMC-PIC Model for Electric Propulsion Plumes

AUTHOR: Dominic Lunde

DATE SUBMITTED: June 2019

COMMITTEE CHAIR: Amelia Greig, Ph.D.
Assistant Professor of Aerospace Engineering

COMMITTEE MEMBER: David Marshall, Ph.D.
Department Chair and Professor of Aerospace Engineering

COMMITTEE MEMBER: Kim Shollenberger, Ph.D.
Professor of Mechanical Engineering

COMMITTEE MEMBER: Robert Martin
Computational Scientist, Air Force Research Laboratory

ABSTRACT

A Homegrown DSMC-PIC Model for Electric Propulsion Plumes

Dominic Lunde

Powering spacecraft with electric propulsion is becoming more common, especially in CubeSat-class satellites. On account of the risk of spacecraft interactions, it is important to have robust analysis and modeling tools of electric propulsion engines, particularly of the plasma plume. The Navier-Stokes equations used in classic continuum computational fluid dynamics do not apply to the rarefied plasma, and therefore another method must be used to model the flow. A good solution is to use the DSMC method, which uses a combination of particle modeling and statistical methods for modeling the simulated molecules.

A DSMC simulation known as SINATRA has been developed with the goal to model electric propulsion plumes. SINATRA uses an octree mesh, is written in C++, and is designed to be expanded by further research. SINATRA has been initially validated through several tests and comparisons to theoretical data and other DSMC models. This thesis examines expanding the functionality of SINATRA to simulate charged particles and make SINATRA a DSMC-PIC hybrid. The electric potential is calculated through a 7-point 3D stencil on the mesh nodes and solved with a Gauss-Seidel solver. It is validated through test cases of charged particles to demonstrate the accuracy and capabilities of the model. An ambipolar diffusion test case is compared to a neutral diffusion case and the electric field is shown to stabilize the diffusion rate. A steady state flow test case shows the simulation is able to stabilize and solve the electric potential for a plume-like scenario. It includes additional features to simplify further research including a comprehensive user manual, industry-standard version control, text file inputs, GUI control, and simple parallelism of the simulation. Com-

pilation and execution are standardized to be simple and platform independent to allow longevity of the code base. Finally, the execution bottlenecks of linking particles to cells and particle moving were removed to reduce the simulation time by 95%.

Dedicated to Justin Meek.

ACKNOWLEDGMENTS

Thanks to:

- Dr. Amelia Greig

  Thank you for being my thesis advisor, for being my teacher, for introducing me to the world of Aerospace Engineering, for trusting me in this thesis, for sending me to Japan, and for pushing me so hard that I finally felt like a Rocket Scientist.

- Dad

  Thank you for always being there for me, for camping at the KOA and showing me Cal Poly, for always supporting my life decisions, for all the fall down hugs, and for giving me the slight nudges I need to get work done. Thank you also to the rest of my family. Mom, thank you for always being there for me, and being willing to put me first if I need it. Janelle, thank you for being a constant source of inspiration to me in aspirations, faith, and enjoyment of life. Shawn, thank you for being stoked about the things that I am stoked about, including me in your life, and for pushing me along on this thesis. Nico, thank you for showing me all the love I would ever need. Also, thank you Emily for being my Cal Poly buddy and for editing this thesis.

- Mac

  Thank you for dealing with scatterbrained Dom, for becoming my friend, for being such a hard worker, and for inspiring me to see the world.

- David

  Thank you for understanding that I was still catching on to the whole Engineering thing during the beginning of thesis, for working so dang hard on this project, and for writing a good thesis so I understand what's happening.

- Dr. Graham Doig

  Thank you for being an incredible boss, for supporting me even though you didn't really know me, for teaching amazing CFD and wind tunnel courses so I would have half an idea how to deal with fluids for this thesis, and for teaching through doing.

- PROVE team

  Thank you for being a family away from home, for having dreams that are way up in the clouds, for giving Dawn your all, and for trusting me with her. Thank you Will for showing me how incredibly a boss can also be a friend. Thank you Thomas and Dave for inspiring me to work hard and dream big. Thank you Kirwan for being my not quite identical twin and for always getting fired up for me. Thank you Natalia for your complete and total love for your friends.

- Joe House

  Thank you for being my family away from home and dealing with your teenage kid Dom learning how to live with peers. Julia, for always having an open ear. James, for always having an open and strong heart. Jess, for taking care of me especially during stitches. Nico, for having such passion for so many things and sharing it with us. Luke, for being so caring and loving to me, and genuinely wanting to know all about my life. Garrett, what are the odds you write one of these? Also thank you to the rest of my Newman community, you made me feel safe.

- Aero Grad

  Thank you to the Aero department and especially the grad students for accepting a physics major into the fold and teaching me your ways. Thank you Brandon for mentoring me and having such a calm and understanding presence as a teacher. Thank you Harrison, Jake, and Matt for being my Aero older brothers. Thank you Daniel for being an awesome friend, and for all those late night rides home. Thank you Lucas for editing this thesis.

- SLOBS

  Thank you SLOBS for being my bros. Thank you for pushing me every practice. Thank you for laughing with me, losing with me, bidding with me, and catching my hucks. Thank you Johnny for being a true friend with a heart of gold.

- Cho-lab

  Thank you Cho-lab for making my stay in Japan one that changed how I view others for the good. Thank you for welcoming me with open arms and dealing with my loud American self. Thank you Takeshi for taking me under your wing but also allowing me to teach you as well. Thank you Hind for being my ride or die friend. Thank you Mark for investing in me as a person. Thank you Victor for coding support and for being genuinely interested and invested in me.

- Michele Da Silva

  Thank you Mickey for always being a super fan of me. From the hours of frisbee nerd talk, hundreds of homework problems you helped me through, and for being a rock of a friend. I can't wait to see how you change the world next.

- Andrew Guenther

  Thank you for uploading this Latex template

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

INTRODUCTION

There are many methods to create something new, but they all begin with an idea that needs to be realized. The path to realization in some cases can involve simply building the final project; however, with aerospace engineering that is not usually the case. In order to achieve the end goal, much planning is needed. An idea is formed, researched, designed, and then built. With many aerospace projects, designing cutting-edge technology requires accurate physical modeling in order confirm the feasibility of a design and examine the effects of changes. Therefore, the field of aerospace simulation and modeling is a large and extensive field. It is one which is constantly changing and expanding as computing power becomes exponentially stronger. The boom in computing power has opened the door to this thesis.

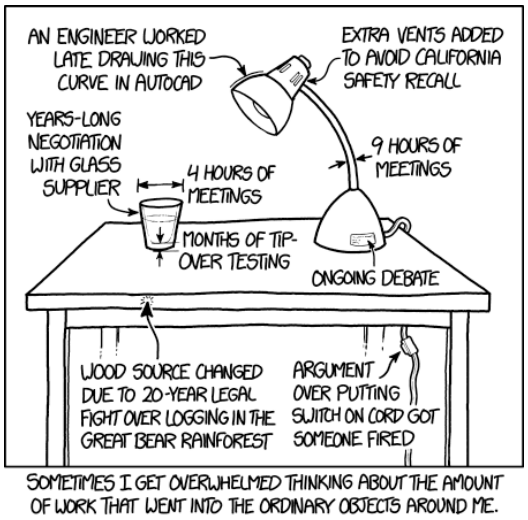

Figure 1.1: Example of various engineering design tasks [1]

1.1 Motivation

Computational Fluid Dynamics (CFD) is the field of study which deals with creating simulations of fluids. This is a critical part of aerospace design and analysis. Viscous fluid dynamics are completely described through the Navier-Stokes Equations.

$$\partial_t u_i + R u_j \partial_j u_i = -\partial_i \rho + \nabla^2 u_i + E_i \tag{1.1}$$

$$\partial_j u_j = 0 \tag{1.2}$$

u_i = Velocity components (m/s)[1]

R = Reynolds number

ρ = Pressure (Pa)

E_i = External Forces (N)

The Navier-Stokes equations, seen in Equations 1.1 and 1.2 in the incompressible form [2], have no known closed-form solution without making simplifying assumptions. They must be solved numerically and, to increase their accuracy, call for increases in computational power to match. CFD is currently an accepted and reliable industry tool for analysis across many disciplines. As computers continue to get stronger, more complicated modeling techniques can be used for unique and diverse scenarios.

A large part of aerospace research involves states of matter where the particle density is low. Atmospheric re-entry of spacecraft, objects in Low Earth Orbit (LEO), planes flying at extremely high altitudes, and interactions with plasma all involve fluids whose particles are relatively far apart from each other. The relative spacing

[1]Throughout this thesis, repeated indices mean sum over that index.

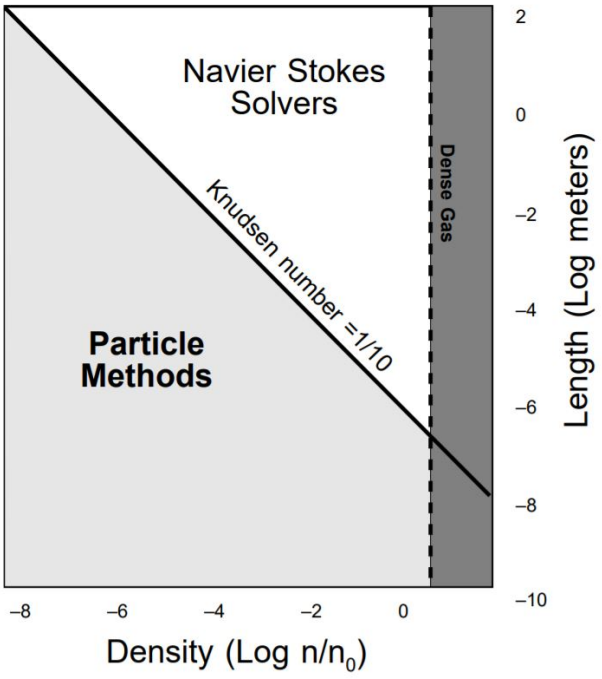

Figure 1.2: Appropriate Simulation Models by Knudsen number [3] n_0 is the density of air at standard temperature and pressure at sea level

between particles can be measured through the Knudsen number, seen in Equation 1.3.

$$Kn = \frac{\lambda}{L} \tag{1.3}$$

Kn = Knudsen number

λ = Mean free path

L = Characteristic length

The mean free path is a measure of how far particles move before interacting with other particles and the characteristic length is a description of the basic length scale of the problem at hand. The ratio of the two (the Knudsen number) shows their relative differences and, therefore, quantifies whether continuum is a good assumption for that fluid. When the Knudsen number is very low ($\sim 10^{-3}$) the particles collide much

more often than they move the characteristic length, which means that a continuum assumption is valid. However, as seen in Figure 1.2, when the Knudsen number is large ($\sim 10^1$ and up) the continuum assumption, and by inference the Navier-Stokes equations, are no longer valid. One reason they are no longer valid is because in the Navier-Stokes Equations (1.2), where the divergence of the velocity is zero, the density of the fluid remains constant, and therefore is incompressible. High Knudsen number fluids are almost always compressible and therefore the divergence of their velocity is non-zero.

There is another equation which can solve high Knudsen number problems. This is the Boltzmann equation, shown in kinematic form for elastic interaction of particles in a plasma by Equation 1.4 [4]. It is based upon the statistical distribution of particles and their momentum.

$$\frac{\partial f_i}{\partial t}+\vec{v_i}\cdot\frac{\partial f_i}{\partial \vec{r}}+\vec{F_i}\cdot\frac{\partial f_i}{\partial \vec{v_i}}=\sum_j\int_0^\infty\int_0^{2\pi}\int_0^\infty[f_i(v_i')f_j(v_j')-f_i(v_i)f_j(v_j)]g_{ij}b\mathrm{d}b\mathrm{d}\epsilon\mathrm{d}v_j \quad (1.4)$$

$f(v)$ = Particle distribution function

i and j = ith and jth species

v = Velocity

F = Force acting on the particle

$'$ = Functions for post-collision

g_{ij} = Initial relative velocity between particles

b = Impact parameter

ϵ = Azimuth angle

The Boltzmann Equation can be solved for a few select unique fluid scenarios. These scenarios usually involve knowing the entirety of the initial state, being in

a near equilibrium state or being in a near vacuum environment [5]. In order to model other more complicated scenarios the Boltzmann Equation is nominally solved numerically, like the Navier-Stokes equations. To solve the equation numerically is a large task, considering the triple integral, the unknown way to find an accurate function for the particle distribution, and the curse of dimensionality. It would follow that a discretion of the Boltzmann Equation would be helpful. This is where a particle-based code begins to make sense.

The most basic algorithm for a particle-based simulation is to keep track of every particle in the domain and calculate each collision depending on the particles' relative distances. While this algorithm, called Molecular Dynamics (MD), is a valid algorithm, computers are not strong enough currently to create meaningful results from such a computationally heavy solution. This is where Direct Simulation Monte Carlo (DSMC) comes into play. DSMC was introduced in 1963 [6], one year before the computer language BASIC was created [7]. Back then, DSMC was seen as an inferior method to continuum solutions on account of the larger processing power required and the apparent abandonment of concrete mathematical solutions [6]. Eventually it was shown that the limit as time-step and grid size approached zero of a DSMC simulation was a solution to the Boltzmann equation [8]. Through the years, DSMC has gained respect and is now is well-known in the fluid modeling community as a standard for high Knudsen number flow simulation methods.

> "The Direct Simulation Monte Carlo, or DSMC, method proved a probabilistic physical simulation of a gas flow by simultaneously following the motion of representative model molecules in a physical space." [8]

DSMC fits into an important gap in fluid simulation methods. It is a particle-based method, which allows it to accurately model high Knudsen number simulations

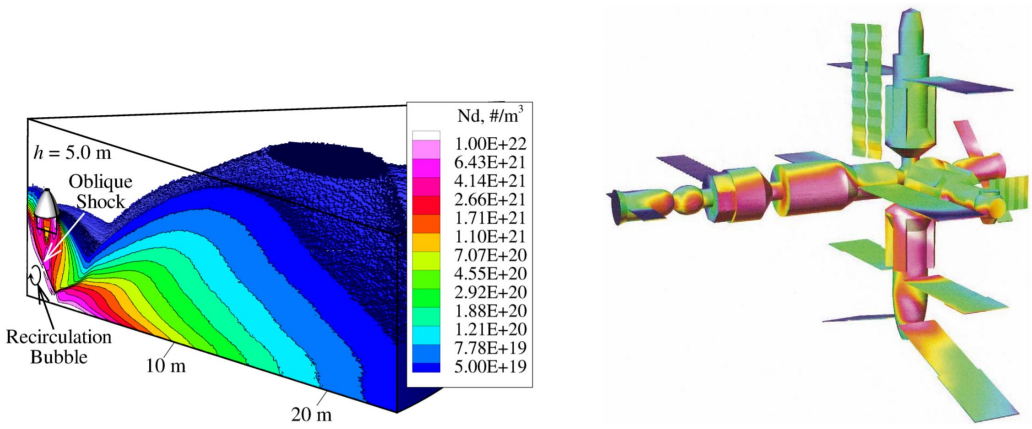

Figure 1.3: DSMC Application Example (left) A rocket lander hovering 5 m above the surface [9]. (right) Predicted surface pressure on Mir from the Space Shuttle RCS during docking [10].

in cases when continuum codes break down. Yet it does this while remaining based on statistical sampling and therefore allows critical computation time-saving mathematics and algorithms to allow DSMC to be on par with legacy continuum methods. DSMC simulates many super-particles in a domain, each of which is comprised of millions of physical particles. DSMC then uses statistics to determine collisions and surface interactions. The super-particles allow for a greater range of high Knudsen number scenarios to be simulated which is of particular interest in astronautics where Knudsen numbers are traditionally high. As seen in Figure 1.3, DSMC is now used for many important aerospace analysis tasks.

Unfortunately, DSMC alone does not account for a large area of aerospace research: plasma. Plasma exists when there are a large number of ions and enough high-energy electrons to keep ionizing the gas. As seen in Table 1.1, plasma can exist in many various situations and environments. Even commonplace things like candle flames or lightning bolts contain some plasma. Two large areas of aerospace research which take place within a plasma environment are atmospheric re-entry and electric propulsion. Both exist in a high Knudsen number environment but include charged

Table 1.1: Various Plasmas [11]. n - Number Density, T - Plasma Temperature

Plasma Type	n (cm^{-3})	T (eV)
Interstellar gas	1	1
Solar Corona	10^3	1
Hot Plasma	10^{14}	10^2
Thermonuclear Plasma	10^{15}	10^4
Laser Plasma	10^{20}	10^2

particles. Therefore, while DSMC would be well-suited for the situation, it is not able to model the charged particles in these situations.

Plasma is comprised, on average, of equal parts of negative electrons and positive ions, otherwise known as quasi-neutrality. However, there can be systematic (electric thruster) or random (solar wind) fluctuations that cause a local net charge. This fact means that the plasma can create electric forces, which in turn affect the charged particles in a plasma. DSMC cannot simulate charged particle flows because it only has mechanisms for collisional velocity updates. DSMC does not have a mechanism to calculate the electric force and update the particles velocities. This is where the Particle-In-Cell (PIC) method applies.

The PIC method is a standard method to calculate the force on electrically charged particles in a particle type model. It is based on a mesh system comprised of nodes, upon which field calculations are made. The mathematics of charged particle forces in terms of PIC will be discussed in Chapter 4. PIC, however, does not account for collisions between particles. To accurately model a plasma for aerospace research, collisions between particles need to be taken into account. This naturally leads to the necessity for a DSMC-PIC hybrid method. A single mesh can be used for calculating collisions and macro-properties through DSMC as well as being used to calculate the fields in the domain and how they affect the particle's velocities with PIC. This allows for a program which can simulate a greater range of scenarios. However, it

may be necessary with more complicated simulations to have different mesh sizes for the DSMC portion from the PIC portions.

One scenario within aerospace research for which a DSMC-PIC simulation is optimal is electric propulsion. There are three main methods of spacecraft propulsion: chemical, cold gas, and electric. Chemical is the classic rocket engines used to put spacecraft and humans into orbit. Cold gas is simply expelling pressurized gas to provide thrust. Electric propulsion uses electricity to create more efficient engines for spacecraft. Electric propulsion can either be electrostatic, electrothermal, or electromagnetic. A nearly universal property of electric thrusters is that the exhaust is usually in a plasma state. An electric propulsion engine, in this case an ion engine, is shown in Figure 1.4. This figure shows the neutral propellant gas being ionized in the chamber, then the ions accelerated out through magnetic grids to create thrust[2]. This results in a plasma plume. Since the spacecraft is in space and the thrusters do not expel dense amounts of gas, this plasma fits right into the realm of a DSMC-PIC simulation. Herein this lies the motivation for this thesis.

1.2 Overarching Goal

This thesis is one of a series of projects. The end goal is to have a Cal Poly homegrown simulation that can simulate an entire electric thruster. While the individual areas of modeling the discharge chamber [14] and modeling the plasma plume [15] are growing strongly, there is a distinct lack of complete electric thruster models in literature. It is technically possible to simulate the entirety of the thruster in DSMC, but the Knudsen number is so low inside of the thruster that computing a DSMC method would be nearly impossible. A simulation of a full electric thruster requires

[2]A more detailed explanation of ion engines and other information about electric propulsion can be found in Reference [12]

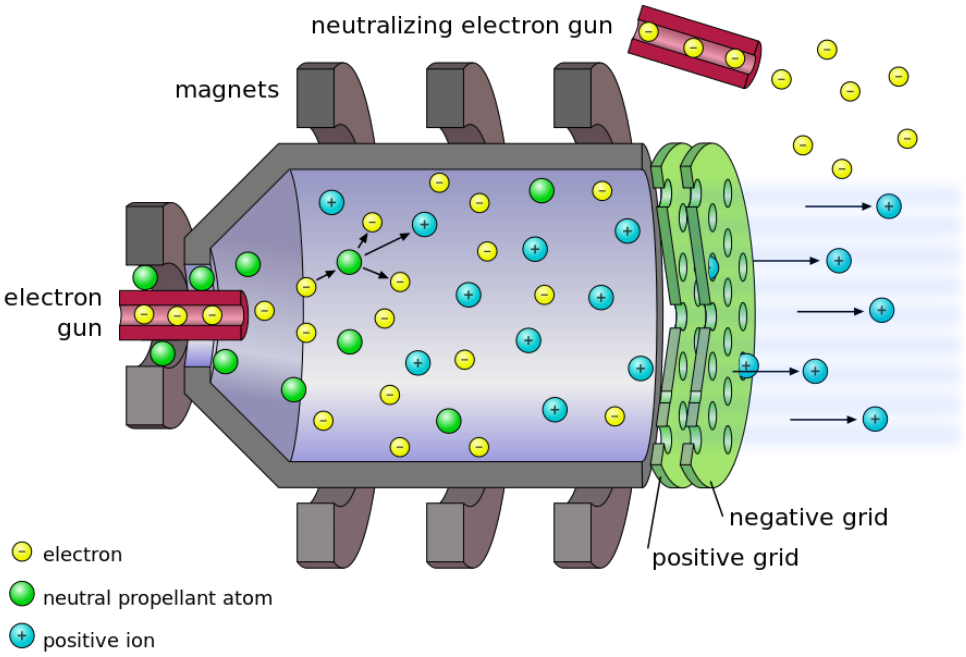

Figure 1.4: Diagram of an Ion Thruster [13]

a continuum simulation for the gas inside the thruster and then a rarefied gas simulation for the exhaust plume where the Knudsen number is higher. In order to model a complete thruster, researchers have compromised by attempting to join together a fluid code and a rarefied gas code. Cal Poly's Aerospace Department is rare in that a fluid code and a rarefied gas code are being developed in the exact same manner from the ground up. Other institutions are also working in this area with success in the initial stages [16, 17]. The early developers and advisors at Cal Poly worked together to establish these codes as being compatible from the beginning.

This thesis is the third project of a homegrown DSMC code, code name SINATRA, which will eventually be able to simulate the thruster's plasma exhaust plume. The first three developers worked together in a staggered capacity to build SINATRA up to a working DSMC-PIC code. The first, Galvez, developed the base framework and kinematics [18]. Next Alliston built up the collisions and particle models [19]. This thesis adds charged particle simulation capability to SINATRA. Concurrently with

this thesis, Gay has been building the fluid side simulation [20]. Intentionally, many properties will be common between the codes. They are both written in C++ and are class-based. They share the same process control items, including the execution style, distribution method, and the other systems items shown in Chapter 3. They share the same mesh type and input class to help them work together as one simulation. A future project will take both simulations and build the interaction system to connect them across each time-step.

Chapter 2

DSMC AND SINATRA

The DSMC method has been refined over years of implementation and testing. A large amount of that progress has been done by Bird, who introduced the DSMC method [21]. Much of the DSMC procedures and techniques discussed here have come from his book which walks a developer through the steps of creating a DSMC simulation [8]. This chapter will explain the DSMC method and the SINATRA's implementation.

2.1 DSMC Overview

At its core, the DSMC method is a particle pusher. It takes a domain, initializes particles in the domain, and, through each time-step, injects, moves, and collides particles. The simple breakdown of a DSMC flow can be seen in Figure 2.1. The large difference between DSMC and a plain particle pusher is that each simulated particle is designed as a clump of actual particles. Therefore, each section and algorithm of the simulation must keep that in mind when calculating physical properties and events. However, for this thesis, a 'super-particle' will be referred to as a particle for simplicity.

The first step is to "Initialize System". There are two critical parts when initializing the simulation. First is the mesh structure. The simulation must know where each cell is in the domain, what shape the cell is, and what type of boundary each of its faces are. There are various algorithms and methods used in generating and storing this mesh. Those ones used for SINATRA are discussed in Section 2.2.2. The

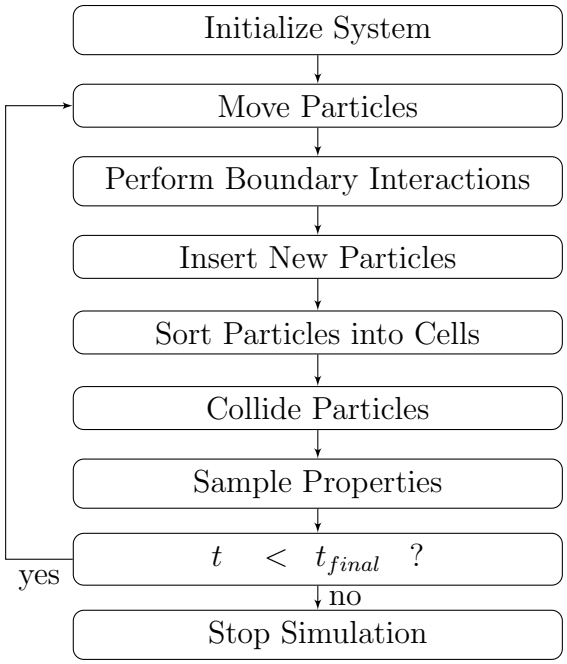

Figure 2.1: Basic DSMC flowchart [18]

second important initialization is of the particles. Importantly, once initialized, the particles should have a uniformly random distribution in their position, as well as a normal distribution around the set domain velocity vector. There are two methods to initialize particles. The first is to randomly insert the number of simulated particles throughout the domain. However, this can cause problems with collision systems and with mesh-particle linking. Therefore, there is a second method called uniform distribution. In this method, the particles are distributed evenly to all of the cells and then within the cell they are randomly distributed. This is a valid method even though it removes some degree of randomness, because injecting and colliding particles re-establishes statistically random behaviour [8, 18, 19]. Therefore, accurate results can be gained in the uniformly distributed method as long as sufficient time-steps are included. The difference between the two systems can be seen in Figure 2.2. Two smaller processes done during the Initialize section are reading information from the user input and linking the particles to the cells they are within.

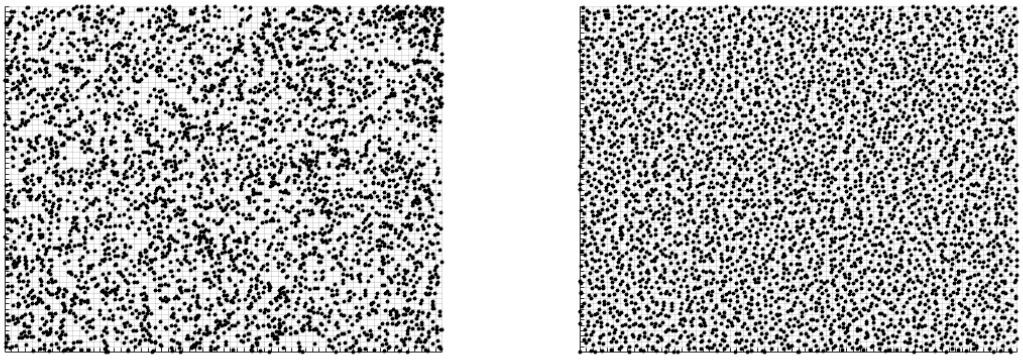

Figure 2.2: Uniform Particle Initialization (left) The generic random initialization of particle position [19]. (right) The uniform distribution of particle position by each cell [19].

The next step is to "Move Particles". This algorithm consists of going through each particle and updating its position using its velocity and the chosen time-step. This is a purely decoupled event and therefore can be split into parallel operations, which has been implemented and is shown in Section 3.2.4.

The next step is to "Perform Boundary Interactions". After moving a particle, it is checked to see if it exited its cell, and if so if it passed through a boundary on the way there. If so, the particle is processed through surface interactions. There are many types of surfaces that can be implemented in a DSMC simulation. A discussion of the ones implemented in this thesis are found in Section 2.2.3.

The next step is to "Insert New Particles". In this stage particles are introduced to the domain through Inlets specified by the user. New particles are randomly inserted in terms of position and velocity, but are inserted with the same distributions required in initialization.

The next step is to "Sort Particles into Cells". Once the particles are all moved, all the new particles are added, the surface interactions are calculated, and the particles which are in new cells are linked back to those cells. This involves looking at nearby

cells and determining within which cell the particle now lies, as well as removing particles no longer in the domain.

The next step is to "Collide Particles". Now collisions are calculated between the particles. This involves using the sphere model for the particle and choosing collision partners through a selection method. A discussion on the models available in SINATRA can be found in Section 2.2.3. These collisions will change the velocity of the particles.

The next step is to "Sample Properties". Once this is completed, properties about the simulation can be sampled. Through sampling during the time-stepping loop, the analyst has the ability to view the simulation over time and determine if it has reached a steady state solution or view the transient or oscillatory nature of the fluid. This allows SINATRA to have the capacity to be a steady state or a transient simulation depending on the scenario and application. The method to sample properties in SINATRA can be found in Section 2.2.4.

These steps continue until the simulation has completed the user specified time-steps. Then the simulation breaks from the loop, performs final data output, and ends the program.

2.2 SINATRA

This thesis centers on being one of the initial development projects of a Cal Poly DSMC code, and the implementation of the PIC component to this code. SINATRA is developed in the C++ language for a few purposes. C++ is a current industry standard for large simulations and therefore has a large support base with libraries, a

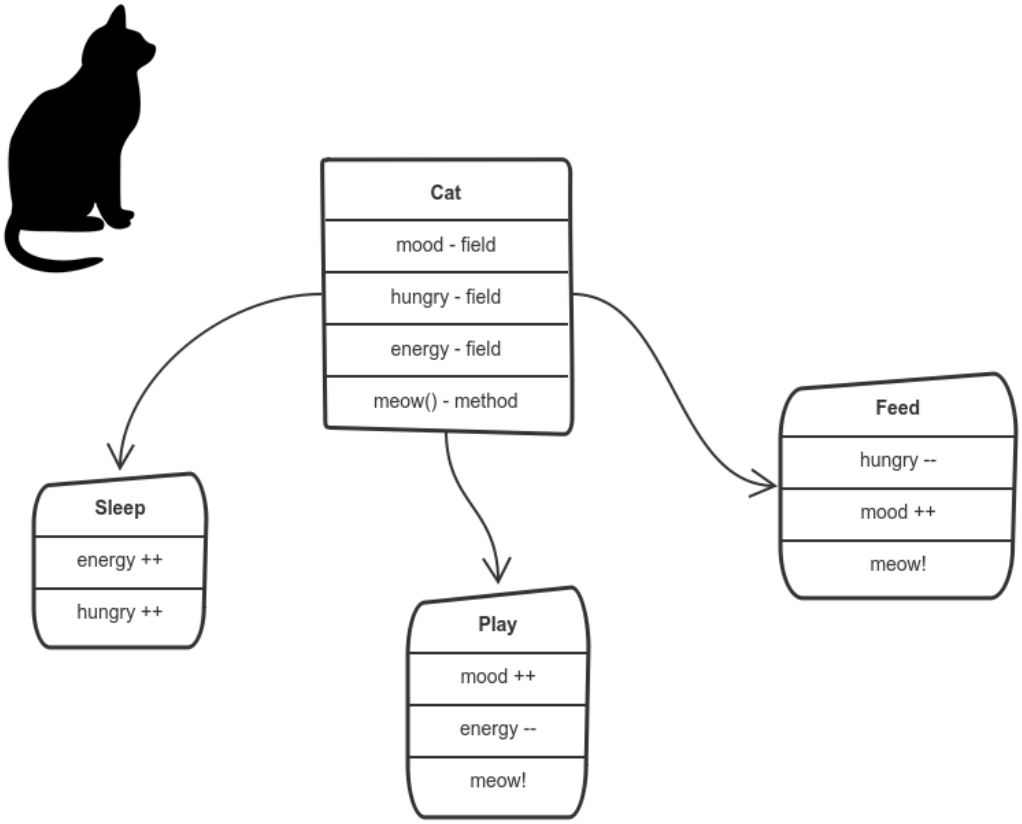

Figure 2.3: A visualization of a Cat Class including the Fields and Methods [22]

developer community, and a large portion of developers know how to read and develop in C++. It is one of the fastest higher level , object-oriented languages.

2.2.1 Object-Oriented

SINATRA is based in object-oriented programming. Object-oriented programming is a method of developing which utilizes 'objects' that can be given variable characteristics. This allows a developer to create an object for a type of item in their program and then create many of those objects each with varying characteristics. It opens the door to functions being associated with various objects and those objects containing

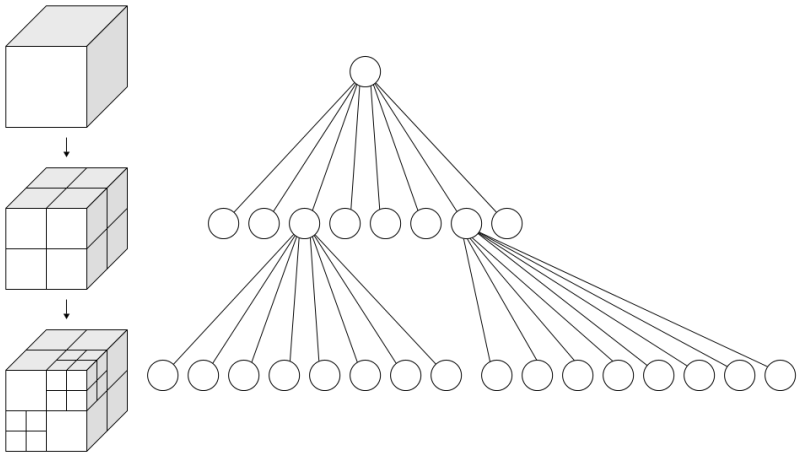

Figure 2.4: A demonstration of an octree mesh and its data structure [23]

an idea of 'self' in referring to the characteristics of that particular instance of the object. A simple version of a class can be seen in Figure 2.3.

This method is particularity helpful in DSMC simulations. There are two large data structures which must be tracked: the mesh and the particles. A class structure fits right into that gap. Refer to Galvez's Thesis to see the exact breakdown of data structures used in SINATRA [18].

2.2.2 Octree Mesh

Another important feature of SINATRA is the octree mesh. Octree is a tree method of storing data, where each parent item has 8 children. An octree mesh has two main benefits; it is a well-known data structure which is well documented and serves as a mesh refinement algorithm. A small example of the octree structure and its refinement capability can be seen in Figure 2.4.

An octree mesh is built from a cube domain, which is cut into 8 initial cells. Then, depending on user settings, more refinement is made until the mesh is well-suited for the simulation case. If the simulation wants to show a curved surface, the cells are

refined near the surface until the mesh is a good approximation of the object. Optimal mesh settings are on a simulation to simulation basis and are a largely a user controlled portion of CFD analysis. However, a discussion on automatic mesh generation and adaptation can be found in Section 5.1.4. The important advantage of octree meshes for DSMC is that when a particle moves out of a cell, it is less computationally intensive to find into which cell it moved compared to more primitive algorithms. The system can move up one level to the original cell's parent and then search all of that parent's children. This algorithm speeds up the linking phase of DSMC.

2.2.3 Models Implemented

There are various ways to physically model particles and domains. A good CFD program will include multiple options for a simulation so the analyst can test different types to see which works best for their simulation case. Therefore, multiple models for boundaries, particle spheres, and collision schemes have been built into SINATRA by Alliston and Galvez [18, 19]. The different types can be seen below. It is also important to note, these models have been built specifically for DSMC, taking into account the 'super-particles' and statistical dependency.

- Boundary Types

  These define the boundary conditions of the walls of the domain.

  - Inflow

    This wall allows particles to be inserted into the domain through the inject particles algorithm.

- Outflow

  The wall allows particles to leave the domain if they cross this boundary.[1]

- Spectral

  This wall is similar to a symmetry plane, the particles are reflected off this wall spectrally.

- Diffuse

  This wall has particles collide with the wall as if there was a user defined imaginary gas on the other side of the wall.

- Periodic

  This wall takes the particle which crosses it and injects it in the same place and position on the other periodic wall, allowing for cyclic simulations.

- Sphere Models

  When modeling particles, it is important to have a model of the electron sphere around the particle and its effective radius.

  - Hard Sphere Model

    This model gives each particle a set radius and cross section.

  - Variable Hard Sphere Model

    This model sets the hard sphere depending on transitional energy.

  - Variable Soft Sphere Model

    This model takes into account the collision partner particle when determining the cross section.

[1]Note, there is also a process by which particles may enter the domain through an outflow wall according to the temperature of the simulated gas on the other side of the wall. This wall will remain named an outflow wall on account of the name within the heritage code. It would be future work to add a new boundary wall which has a pure outflow wall and to rename the current one to accommodate.

- Collision Schemes

  There are many different ways to choose particles for collisions and to simulate those collisions. Note - it is also possible in SINATRA to turn off collisions.

  - Discrete Sub-Cell

    This method cuts each cell into smaller sub-cells, then chooses particles from the sub cell to collide using an accept-reject method.

  - Nearest Neighbor Scheme

    This method checks the distance between every particle in the cell and chooses the closest particles as the collision partner.

  - Trajectory Scheme

    This method takes the trajectory of the particle in question, creates a virtual sub-cell from its projected position, and then chooses a collision partner from that.

2.2.4 Output Methods

Finally, an important part of a DSMC simulation is outputting the properties of the flow. SINATRA has three main methods for outputting information. The first is the simple time-step file. This file is appended to after SINATRA completes each time-step. This allows the simulation to be tracked as it is being run in the background. The next method is the main macro-properties sampler. It goes through all the sample cells, which can either be the lowest or second lowest level of cells and performs two loops. First it looks at all the particles in that cell and sums their properties, then it goes through all the species and calculates the various averages and data for which the specific output case calls. These values are output in a TecPlot^{TM} format and can be adjusted to be printed out every time-step or every few steps. The final output

method prints out each particle location and velocity. This creates a strong debugging tool to understand exactly what is happening in the flow.

Chapter 3

SYSTEMS ENGINEERING

All large aerospace projects can be viewed a group of integrated systems. The systems must be designed and managed in order to provide a reliable and efficient final product. Systems Engineering is the discipline which covers this area. SINATRA is a complex project and program which needs a smooth integration of the various program and sub-systems. This chapter will explain the various systems engineering implementations in SINATRA including documentation, workflow, and user interfaces.

3.1 Documentation

In order for SINATRA to be used by other users and developers, the code must be well-documented. Accordingly, systems of documentation are being put in place for all of the different levels of instructions, guidelines, comments, and information. The documentation can be broken into two sections; for the user and for the developer. These documentation systems must be simple, reliable, clear, and resilient, in order to make the code easy to distribute, simple to learn to the level necessary, and for changes, bugs, and suggestions to be centralized.

3.1.1 GitHub

The system chosen to organize and host SINATRA for developers is GitHub®. GitHub® is an online file storage, syncing, and collaboration work-space for developing code

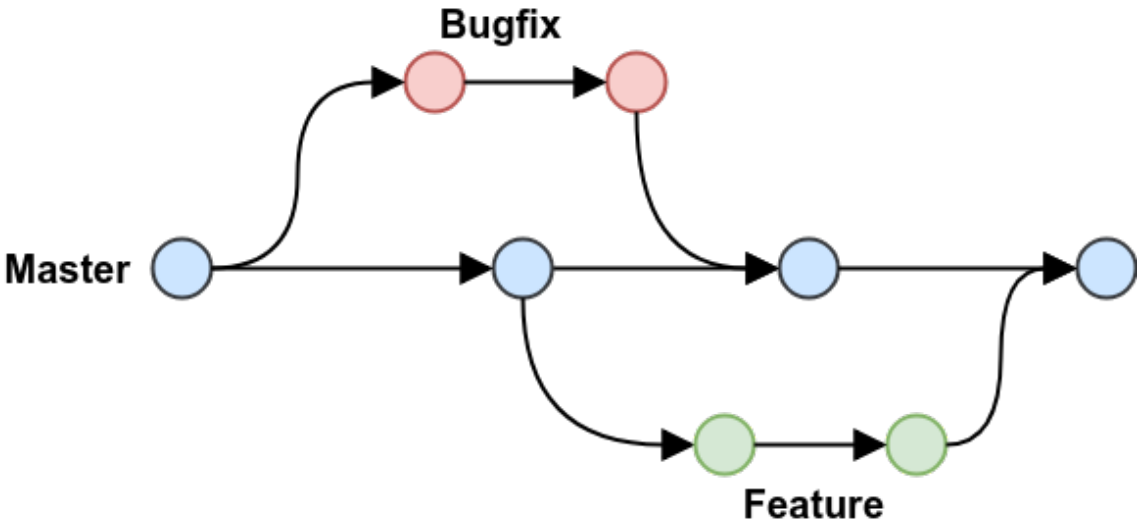

Figure 3.1: Example of GitHub® workflow [24]

bases. It is popular and easy to use such that support for its use is strong. SINATRA is housed on GitHub® by the Cal Poly Aerospace Department. On account of the code's license, the repository is private. Developers can get access by contacting the Aerospace Department at Cal Poly, SLO.

GitHub® has three major features used in SINATRA: the commits system, the branches system, and the ReadME files. The commits system is a method of allowing multiple developers to work on the same code base without breaking the other developers builds. Each developer works on the code on their local machines and once they have a stable addition to SINATRA, they commit it to the GitHub® repository where it is merged into the current version. Whenever other users are working, they can pull those changes on to their local machine. This helps development move along smoothly, although it does require communication between the developers to ensure they are not working on the same lines of code and creating different outcomes. This commit structure also allows the code base to be version controlled, which helps with mistakes, reverting to older versions, and following the change logs.

Another feature of GitHub® utilized during the development of SINATRA was the branches feature. Branches allow a user to create a separate branch of the code base. An example workflow can be seen in Figure 3.1. As shown, there is a master branch which holds the current official version. When a developer wants to build a new feature, they create a branch of the code. On that branch they develop the feature so that the master can continue to be used for official uses. Once the feature is complete the developer submits a pull request which shows the feature and resulting changes to the master branch. The team then approves the feature and it is merged into the official master branch. Branches are used to develop a large new section of code with the features and security of GitHub® and without cluttering the team's main code with testing and validation edits. This process allows developers to command their own section of the code, update and test, and then make it available for the other developers to use. This ensures a constant workflow where multiple developers can work on different features without having to worry about whether their testing and tinkering will hinder others' work on the master branch. In this way, the development continues without bottlenecks, or need for constant communication between the developers.

The final feature used in GitHub® were the ReadME files. These files are automatically displayed by GitHub® when the directory is entered. This allows the developers to convey how a directory fits into the code base, specifics on the files in the directory, and instructions on how to use the directory. The author has outfitted all of SINATRA's directories with ReadME files. These ReadME files give specific directions on how to use each repository of the code base and how to preform basic functions. These features were the reason that the author choose GitHub® to host SINATRA, and they were utilized during development in order to allow efficient and clear code creation. SINATRA's main ReadME page is shown in Figure 3.2, as shown on the GitHub® website.

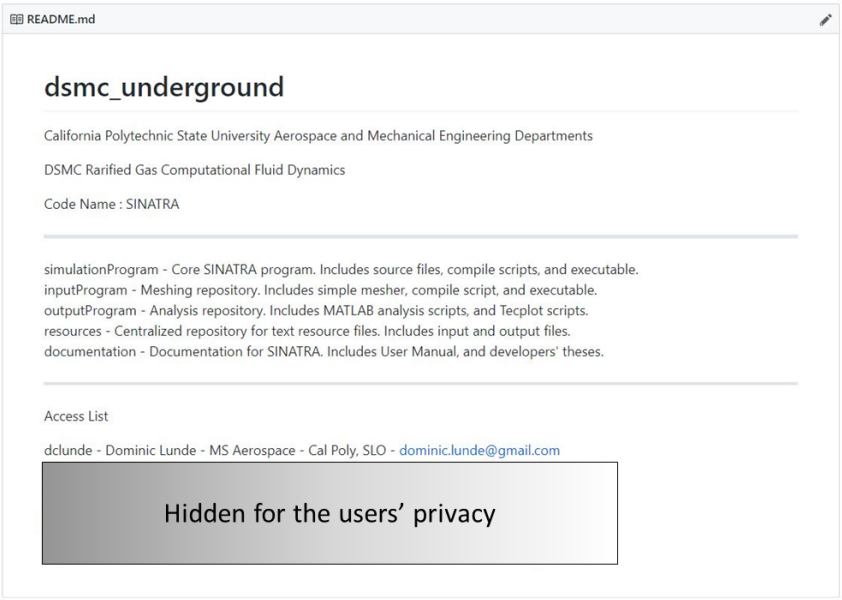

Figure 3.2: SINATRA's main ReadME page

3.1.2 Doxygen

Doxygen is an automatic documentation creator from source code.[1] It was chosen, configured, and used to create the SINATRA Developer Manual. Doxygen is given access to the source code, which it searches through and creates comprehensive documentation of the code base. For SINATRA, it creates descriptions of all of the classes, functions, and files. It shows what each class consists of. For example, it shows that the Mesh class contains structure classes, public types, public member functions and more as seen in Figure 3.3. It is built in HTML, so each attribute is linked to the actual function. It is also possible to see the location of that item in the source code.

The important part of Doxygen is that it allows the user to customize the documentation. It allows the HTML file to be built in many different ways to make it work best for SINATRA. But more importantly, it takes comments made in the source code about the attributes and displays them in a clear and concise way. This

[1]Doxygen: Main Page - `http://www.doxygen.nl/`

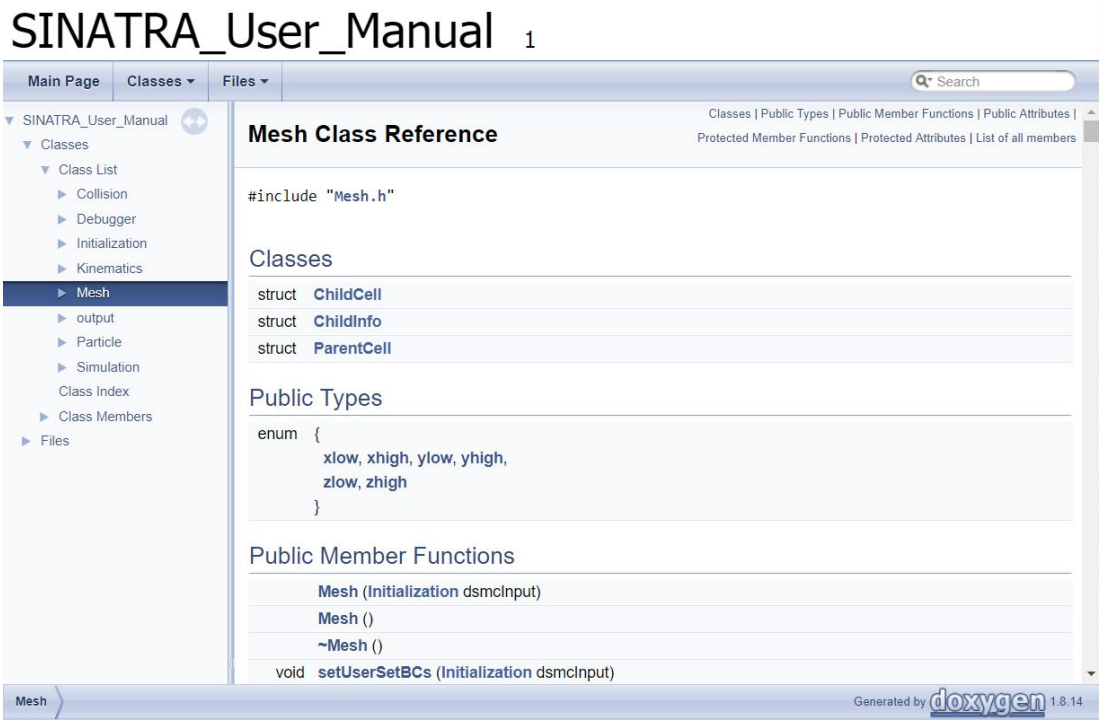

Figure 3.3: Documentation created by Doxygen for the Mesh Class

allows a developer to quickly find an attribute where it is referenced in the rest of the source code, where it is built, and what the creator commented about it. This allows new developers to quickly learn SINATRA and start developing their own features. It also allows quick debugging of heritage code by new developers, which ensures that they will not run into an error that can only be reasonably fixed by the original developer. The author has created a Doxygen manual of SINATRA, as well as a Doxygen input file and batch script for easily updating the manual as more developers add to SINATRA. This was completed by first learning Doxygen's documentation and running test cases. Then applying to SINATRA in a non-intrusive manner with a simple .bat script so the users can view the manual without interfering with the source code. See Appendix B for a list of all classes and files in SINATRA which have been commented with Doxygen. This is not a comprehensive guide to exactly what each function and variable does, but the author has commented every file and class in SINATRA's source code to guide new developers.

3.1.3 User Distribution

As mentioned above, GitHub® will be the primary source of distribution for developers of SINATRA. However, for less serious users, there is a simple distribution. It includes the mesh and simulation executables and input files. It also includes the GUI executable and finally a simple MATLAB® analysis script to visualize the particles. It has documentation which guides the user through a simple test case and show the user how the parts work together. This walk through can be found in Appendix C. This distribution is packaged up in a zip file. The executables and other parts will only work on a Windows computer. Mac and Linux capability can be built through the source code and compiled on a Mac OS and Linux machine respectively.

This zip file can be sent to students for them to try simple test cases and produce new results. This distribution, while very simple and light in terms of file count, is actually a nearly full distribution of SINATRA. Almost all functions of the SINATRA work without the rest of the file structure and other code, therefore, it can be used for things such as uploading to a more powerful computer to ensure the correct version of the code is being used. This allows a more direct way of controlling the simulation in new environments; however, it does exclude much of the functionality found within the actual code base itself. There are many test cases and options which can be activated in the source code. This distribution can be used to share SINATRA with relevant parties who want to produce DSMC results.

3.2 Workflow

There are three main tasks when using CFD: meshing, simulation, and analysis. DSMC-PIC is a subsection of CFD; therefore, the original developers have designed

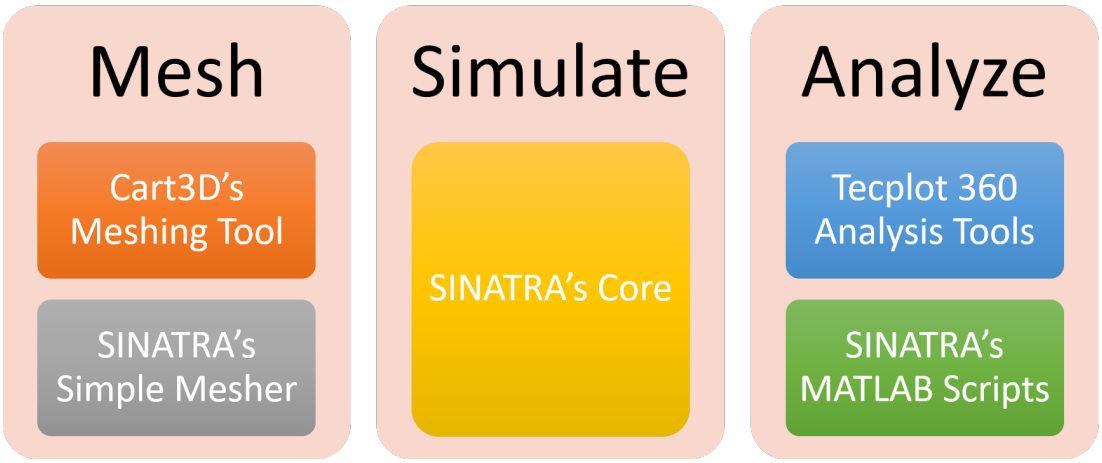

Figure 3.4: Workflow System for SINATRA

systems for all three sections. However, most CFD analysis does not require the user to do all three tasks each time. A single mesh can produce many different simulations, and there can be many different analysis tasks from the same simulation. Therefore, SINATRA was broken down into three parts according to the three tasks it is able to accomplish. Each of those sections have their own repository, source code, and executables. They also share a resources repository between them. This allows the user to work in the meshing system and create the mesh or meshes necessary for their task. Then they move to the simulation system and use the newly created meshes as part of the simulation input. Once they have simulated the domain, they can take the created output files and run different analysis codes or create their own for their specific task.

3.2.1 Mesh

The original developers have designed this workflow for SINATRA while including the option for third party software to be used for the meshing and analysis sections. As seen in Figure 3.4, Cart3D™ [2] was chosen to be the meshing 3rd party additional

[2]Cart3D Documentation - `http://www.nanda.org/`

tool. Meshing a domain for SINATRA requires an octree mesh. This capability is available within native SINATRA itself. The homegrown meshing tool is able to create an octree mesh with any user specified resolution. When an object is added to the domain, meshing becomes a much more complicated task (outside the scope and purpose of SINATRA). Cart3D^{TM} is a CFD analysis tool by NASA which is available to universities. It has within it a three-dimensional octree meshing tool, which can take geometries and domain sizing as inputs. SINATRA will take the outputted mesh and use it for simulations. Cart3D has not been tested with SINATRA. It has been slated for future work for another developer to fully integrate Cart3D^{TM} and geometry boundaries with SINATRA[3]. This thesis uses the homegrown meshing tool exclusively.

3.2.2 Simulation

SINATRA has been designed to be simple to develop and execute. Execution is completed through one executable file and one input text file. For simplicity, Windows^{TM} users can drag the .txt file onto the .exe file to run the simulation. It can also be run through the command line with the path to an input file as the command line argument, or if the argument is missing SINATRA will request the user to input the filepath. The executable and output do not depend on using a specific Integrated Developer Environment like Visual Studio^{TM} or even using a certain operating system A user can run many simulations or string together meshing, simulating, and analysis simply through a batch script. An example script is shown in Appendix A.

SINATRA was also deliberately built to be machine independent to reduce the risk of the code not being developed further or used for new tasks. This was accomplished

[3]It may be necessary to run Cart3D within a Linux^{TM} virtual box for Windows^{TM} users

through making compilation very simple. It requires only a single command with no additional libraries[4]. There are sample compile statements in the ReadME and compile scripts, but even an intermediate C++ developer could figure out how to compile SINATRA from the file list alone. This helps new developers move quickly through the code learning phase and can even allow beginners to explore the code base and test more complicated features. SINATRA has been tested through being compiled with various compilers and on different operating systems.

3.2.3 Analysis

After the simulation phase is completed, the user can use the SINATRA output files for analysis. The original developers created MATLAB® scripts within SINATRA to perform basic types of analysis. For other analysis, Tecplot 360™ [5] has been chosen as a third party tool. Tecplot™ is a specific CFD analysis and visualization tool, which can show the mesh, geometry, and fluid flow. It includes robust visualization and animation tools as well as various analysis functions. SINATRA can output data in a format that Tecplot™ reads natively. Tecplot™ and SINATRA's integration has been tested and used by the first developers.

SINATRA's analysis section has not been built with an encompassing set of features to complete any task. It is up to the future users to determine the analysis they need to accomplish, edit SINATRA's output class to accommodate, and compile the output data into the format best suited the situation. This can be completed through looking at SINATRA's output class and reformatting other analysis techniques for the task at hand. Tecplot™ and the included MATLAB® scripts can do a majority

[4]Need OpenMP™ for parallelization

[5]Tecplot 360 CFD post processing tools to analyzedata - `https://www.tecplot.com/products/tecplot-360/`

of the beginning analysis, but the most detailed tools will need to be built by new developers.

3.2.4 Execution Time

A DSMC code is by nature a very computationally intense program. It requires a large amount of memory to store all of the data of each particle and mesh cell. It requires a lot of computational power to calculate the movement and collisions, therefore, it is important to design the code to be efficient and powerful. This is why C++ and classes were chosen for SINATRA; however, it is still a slow simulation for a large meshes with high particle densities. It is slated as future work for a developer who specializes in computer science and computational optimization to reduce the execution time of large SINATRA runs. However, there are a few bottlenecks which were removed by the author. This improvement helps manage simulation time, especially when the Poisson equation solver is included.

The simplest and most effective way to reduce simulation time on a DSMC simulation is parallelization. Parallelization is a complex and involved field with many competing ideas on best practices. There are many discussions about best ways to parallelize DSMC codes and PIC codes. Parallelization itself is also on the forefront of new technology at the time of this thesis. Moore's law has allowed programmers to have a large amount of memory for their simulation, so that is rarely the constricting factor. Processors seem to be approaching an asymptote in terms of their power for user made systems on languages like C++. Breaking the simulation between multiple cores or even within the graphics processing unit (GPU) seems to be the new normal for decreasing execution times. For SINATRA, there are many parallelization possibilities. It is slated for future work for another developer to optimize the parallelization capacity. At this time, simple parallelization has been developed by the

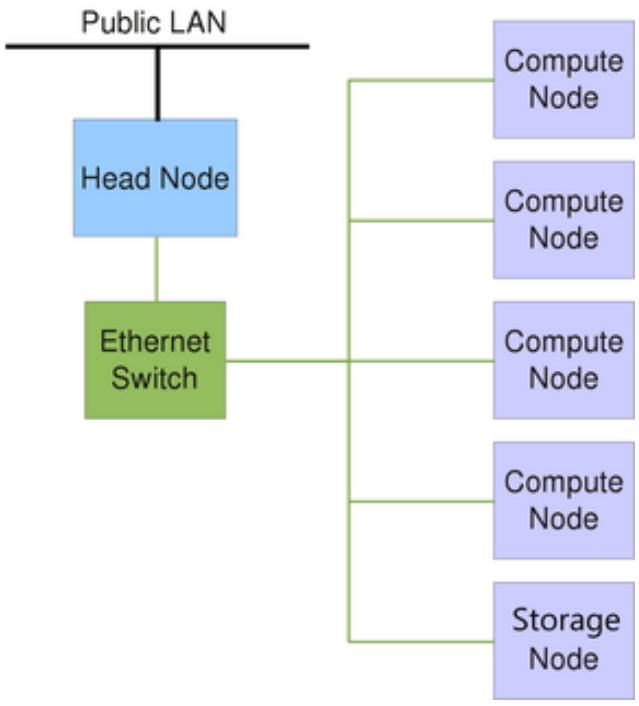

Figure 3.5: System Architecture of Cal Poly HPC Cluster [25]

author. During the particle propagation phase, there is no interaction between the commands, therefore, it can be parallelized by using the library OpenMP™ [6]. This was completed through first identifying the loop in the simulation which moves each particle. Then the loop is reformatted to remove dependencies on any variables which must be updated sequentially. This allows the simulation to run in many different cores without interfering with the other cores. Finally, debugging is preformed to find any errors like segmentation faults and pointer errors. Parallization has been implemented into the input file. It can be enabled through an optional keyword in the input file and ensuring the library is included in the compile statement.

The Aerospace Department at Cal Poly, SLO owns a server that is available to aerospace students and faculty. It is referred to as Bishop High Performing Cluster [25]. It includes a workload manager so that each user only needs to submit their

[6]Home - OpenMP - `https://www.openmp.org/`

jobs and they are separated through the different nodes on the cluster. The cluster allows each full user 48 cores and 64 GB of Ram [25]. This is critical to SINATRA. Through the parallelization the simulation can be run in parallel on the server. Not only does Bishop have quick processors, but it also has the 48 cores between which the simulation can be broken up. This enables the execution time of a large simulation to be cut down by significant amounts, seen in Table 3.1. This server also allows users to run simulations off of their local machines, which helps compartmentalize heavy simulation runs from everyday tasks. It also allows scheduling of multiple simulations such that large analysis tasks can be completed without human interface. SINATRA has been made with the Bishop cluster as an expected resource.

Another simple process to reduce the execution time is during the linking phase. After the particles are created, they must be associated with the cells that they are in. To do this it requires a nested for loop through all particles inside all cells. This is a very slow process. In order to increase the speed, it is possible to allow the linking process to ignore the particles it has already linked and ones which aren't in the domain. This has been been implemented by the author, however, there is a simpler solution. There is an option in SINATRA to seed particles in a uniformly random way [18]. This method creates the same number of particles in each cell but randomly distributes them within the cell. With uniform initialization, the linking process can be removed completely, and the execution time is significantly reduced. This has been implemented by the author. This was completed through reformatting how particles are created in SINATRA. A new constructor for particles was created which includes the index of the cell which it was created in as well as the index of the next particle for the particle chain. The other constructors for particles are updated so that they explicitly state that they are not yet linked to a cell and don't know their next particle. These are two main things are what are added to the particle object during linking. Then during initialization, the cells are given the information of their

Table 3.1: Execution Time Comparisons

	First Iteration	Time-step Average	Total Time
Parallel and No Linking	9 sec	10 sec	1 min, 38 sec
Serial and No Linking	3 min, 23 sec	3 min, 23 sec	33 min, 50 sec
Serial and Linking	15 min, 1 sec	3 min, 58 sec	50 min, 47 sec

first particle and the total number of particles in this cell. These two things are what are added to the cells during linking. These two changes allow linking to be removed for the uniform initialization case.

A simple timing comparison was completed to examine the effectiveness of two execution time reduction methods. A collision-less simulation with all diffuse walls was run on the HPC server. The simulation used 32768 Cells, 5×10^{19} real to simulated particles, 1×10^{26} number density which gives 2 million simulation particles in the domain, and 10 time-steps of 1×10^{-8} seconds with 0 m/s stream velocity and 300 Kelvin temperature. As seen in Table 3.1, including these optimization techniques has reduced the simulation time by 49 minutes for this simulation. By removing the linking process, we were able to save a large amount of time during the set up[7]. Also, by adding the simple parallelization we were able to reduce the simulation time by 95%, from over 50 minutes to under 2 minutes. These two techniques combined make an appreciable difference in the execution time. They allow more complicated simulations to be run within the same amount of computation time. This allows the user to run longer simulations or multiple in order to increase confidence in the accuracy of the solution, as well as reduce the variance caused by the randomness in a DSMC simulation.

[7]There is a slight difference in the iteration time for the linking and no linking runs because the no linking run did not use the uniform initialization because uniform initialization has has the linking section bypassed in the current version.

Finally, an important tool for execution time analysis is debugging and profiling. It is important while developing software to have the ability to debug the code to see exactly what it is doing. This is usually achieved through the Integrated development environment (IDE); however, in an effort to make SINATRA IDE independent, the code has built in a method which allows the GNU Project Debugger to be used on the code[8]. This is a command line interface for g++ compiled executables. It is a legacy debugger that is well documented and is included in most installations of the g++ compiler. Using gdb on SINATRA allowed the author to solve problems within various improvements including removing linking and adding parallelized code. Secondly, a profiler allows the user to view exactly how often each line of code is run, how long it takes to run, and the sub processes, which contributes to that execution time. This is a strong way to identify simple coding bottlenecks and optimizing the coding time. The author has set up the compiling and code base in order to fit with the profilier Very Sleepy™ [9]. This was completed by determining the version of g++ which the profiler required, compiling in that method, and then allowing the profiler to access the executable. Very Sleepy™ is a light and simple profiler which shows each line of the source code and the timing involved. By setting up SINATRA to be widely compatible it allowed these developing tools to push SINATRA towards an industry standard level.

3.3 User Interfaces

The user interface is an aspect to distributing a new code base. The interface must be simple, but powerful. For SINATRA, the author chose a simple text input file as the base user interface. This combined with a simple execute statement and an

[8]GDB: The GNU Project Debugger - `https://www.gnu.org/software/gdb/`

[9]Very Sleepy documentation - `http://www.codersnotes.com/sleepy/`

accompanying GUI allows for a range of users to use SINATRA to accomplish their tasks.

3.3.1 Running SINATRA

SINATRA is controlled through a text input file. The developers compile and distribute a new version of SINATRA. This distribution is at its core a .exe file. Also included is an example text input file. The most basic user only needs to drag the input file onto the .exe file (in Windows^{TM}). They can also just run the executable and input the text input filename or input it as an argument while running the executable in the command line. An example is shown below.

```
>>DSMC.exe Complex_Input.txt
```

These multiple options allow the end user to run SINATRA without being caught up in the one specific way of running the program. This ensures that the program will not fall to a classic hereditary code problem where only the original developers are able to run and use the simulation.

3.3.2 Input Files

The input file utilizes a two-line system. The first line is a trigger word and the line immediately after it contains the input for SINATRA. The input file is broken into 5 parts. First, is the constant header input. This includes the basic information that includes time-step, number density, mesh filename, and velocity. Next are the boundary conditions which are broken into each of the 6 boundaries. Next are the species information, then the output files and frequency. Finally, the optional keywords complete the file . Each section is split by three stars. This system allows the

entire simulation to be compiled once, and complete various tasks just by editing the text file and using it as input. The input file guide can be found in Appendix D.

SINATRA is not optimized as an open source and commonly distributed system. It is made to be a Cal Poly homegrown code base; therefore, it is not built with airtight error checking, complete documentation and support, or perfect distribution and releases. The goal is to have enough of each of those components that competent users in the Cal Poly community may be able to use it to complete simulation and analysis tasks. It is possible that there are bugs within the input system in which it will seem that there was no error and that results are displayed, but they are not physical at all. There are error management functions and systems, but they are not foolproof. The user should read previous theses, understand the input file, and have an understanding of DSMC. That being said, it is possible to help the user understand these things simply through a clickable interface.

3.3.3 Graphical User Interface

The main goal of SINATRA is to be able to simulate rarefied gas for research purposes. It is aimed for use by faculty, graduate students, and specialized clubs in focused research and projects, however, a DSMC-PIC simulation is a useful tool for even undergraduate classes to test and work with. With that purpose in mind, a Graphical User Interface (GUI) for SINATRA has been created. This GUI is built using the MATLAB® GUIDE tool. It allows a user who has never seen the source code to be able to set up and run simulations. It is built with the expectation that the user does not know how to set up a SINATRA run. The default set up of the GUI can be seen in Figure 3.6.

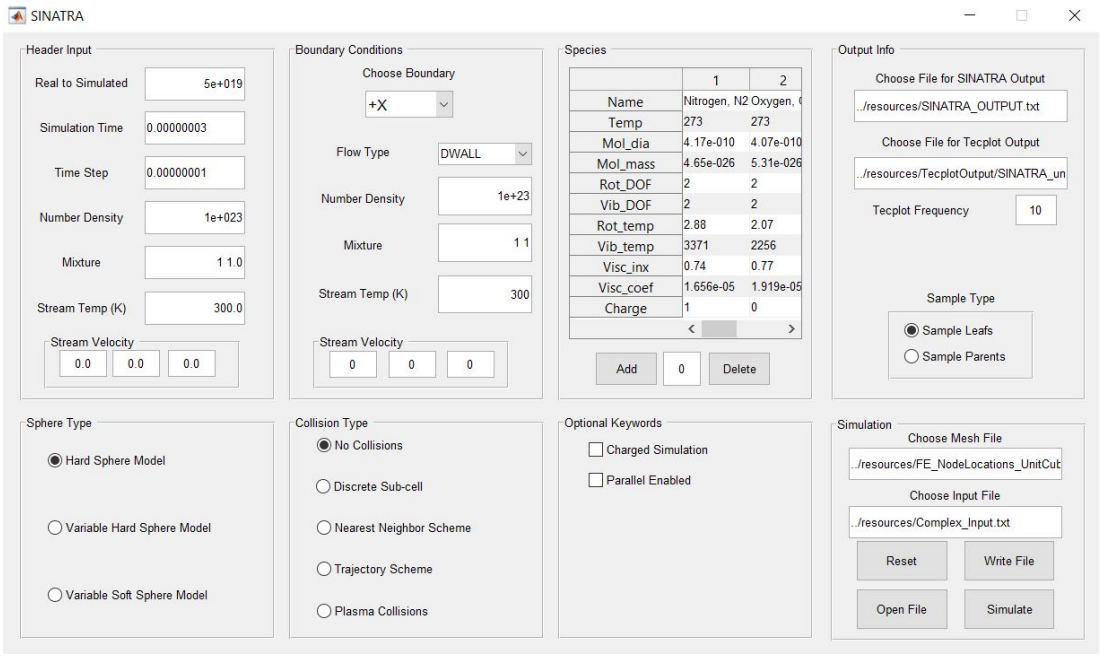

Figure 3.6: Default Setup for SINATRA GUI

The GUI allows the user to input the various parameters of a SINATRA run. It does low level error checking on their input to help the user make simulations without many errors. Then the user can create an input text file through a click of a button. The simulation converts the user's inputs into the format of SINATRA's input file. Then the user can select 'Run' and the GUI will take a current distribution of SINATRA's executable and run it with the recently created input file. The SINATRA GUI is also available as a pure executable or MATLAB® figure, so that the end user does not have to run complicated code.

Chapter 4

CHARGED PARTICLES

A DSMC simulation has many applications in aerospace engineering. However, it does not capture the full picture when gasses are composes of ions and electrons. Plasma, which is composed of charged particles, needs the electrostatics to be taken into account in the simulation in order to create an accurate model. This chapter will explain how to simulate charged particles with the particle-in-cell method, how that method was implemented in SINATRA, and how the implementation was validated.

4.1 Particle-In-Cell

In order to model the charged particle interactions, a particle-in-cell method is implemented in SINATRA. As previously stated, the plasma is quasi-neutral in that it has nearly even densities of ions and electrons. There can, however, be patches of charge inequality. The gradient in the charge density creates an electric field, which in turn applies a force on the particles and changes their velocities.

The characteristic length of those charge inequalities can be calculated through the Debye Length, λ_{De}, shown in Equation 4.1 [26]. This Debye length is the first plasma property which helps with setting up the simulation. The Debye length gives us a good approximation for cell size when creating a PIC mesh.

$$\lambda_{De} = \sqrt{\frac{\epsilon_0 \, T_e}{e \, n_e}} \tag{4.1}$$

ϵ_0 = Permittivity of free space

T_e = Electron Temperature

e = Elementary charge

n_e = Electron density

Similar to the Molecular Dynamics method, it is possible to model every particle in a domain[1]. Charged interactions between particles would be calculated through the Coulomb Force, seen in Equation 4.2. The Coulomb Force is only the force between two particles. Therefore, to calculate the force on a single particle, the force of each particle in the domain must be summed. This is still computationally infeasible.

$$\vec{F_C} = \frac{1}{4\pi\epsilon_0} \frac{q_1 q_2}{r^2} \, \vec{r}_{12} \tag{4.2}$$

F_C = Coulomb Force between two particles

ϵ_0 = Permittivity of Free Space

q = Charge of a particle

r = Distance between particle 1 and 2

$\vec{r}_{12}$ = The direction of the line connecting the two particles

In PIC, the particles move according to the Lorentz force, seen in Equation 4.3. This force can be calculated across the entire domain and then applied to each particle, reducing the order of the simulation to $O(n)$ where n is the number of particles in the domain [27].

[1]This explination of the Electro-Static PIC method draws much of its material from [27] because it was used as a reference and it is attempting to explain the same method at the same level as this thesis.

$$\vec{F} = q(\vec{E} + \vec{v} \times \vec{B}) \tag{4.3}$$

q = Particle Charge

E = Electric Field

v = Particle Velocity

B = Magnetic Field

It is not in the scope of this thesis to examine the effect of the self-induced or an applied magnetic field, therefore the force is reduced to $\vec{F} = q\vec{E}$. The electric field can be calculated through the gradient of the electric potential, as shown in Equation 4.4.

$$\vec{E} = -\nabla\phi \tag{4.4}$$

ϕ = Electric Potential

The electric potential comes from the double gradient of the charge density. The electric potential can be seen in Equation 4.5, which is also known as the Poisson Equation, specifically for electrostatics.

$$\nabla^2\phi = -\frac{\rho}{\epsilon_0} \tag{4.5}$$

ρ = Charge Density

ϵ_0 = Permittivity of Free Space

The charge density is found using Equation 4.6. More information on the PIC method to calculate the charge density is given in Section 4.2.

$$\rho = e(Z_i n_i - n_e) \tag{4.6}$$

e = Elementary Charge

Z_i = Ion Charge

n = Number density

i = subscript for ions

X_e = subscript for electrons

An important assumption is made for this thesis. Electrons are much smaller than ions and therefore travel much faster. The difference in speeds between the two means that a simulation which includes both ions and electrons as simulated particles would need to have a time-step which captured the electron movement, and therefore would be wasting a lot of time on the ions barely moving. In order to simplify the simulation, a fluid assumption is made for the electrons. This is a valid assumption when viewed from the ions reference frame because the electron particles themselves move so fast that the ions are only affected by the overall distribution of electrons. This fluid assumption is calculated through the Boltzmann relationship [27], and can be seen in Equation 4.7.

$$n_e = n_0 \exp\left(\frac{\phi - \phi_0}{kT_e}\right) \tag{4.7}$$

n = number density

ϕ = Electric Potential

ϕ_0 = Initial Electric Potential

k = Boltzmann Constant

T_e = Reference temperature of the electrons

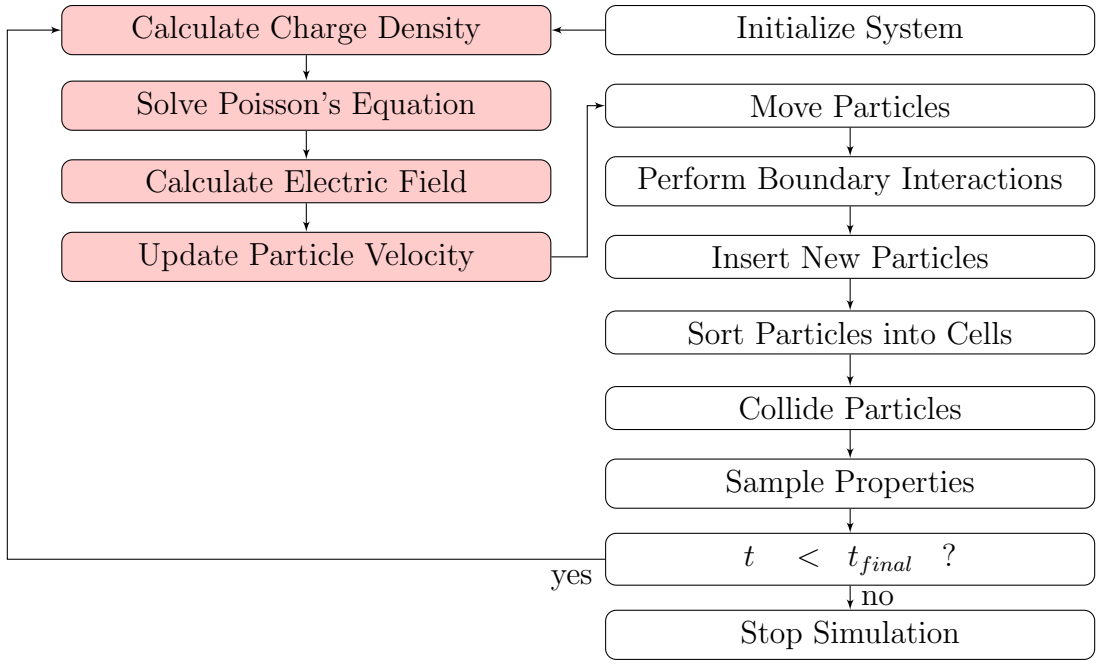

Figure 4.1: DSMC-PIC Hybrid Code Flow. White blocks are DSMC algorithms [18]. Red blocks are for PIC.

4.2 Particle-In-Cell Algorithm

This set of equations then gives us the particle-in-cell method; firstly, the charge density is calculated, then the electric potential, and finally the electric field. The electric field is then used to update the velocities of the particles. The addition of these calculations into the DSMC method is shown in Figure 4.1.

There are many ways to take these formulas and code flow and turn into a working simulation. There could be multiple methods for all of these calculations, which leads to the various techniques for PIC codes, similar to the various DSMC codes. The ones chosen for SINATRA are simple and robust. First, while the DSMC method is based upon cells full of particles, the PIC method depends on the nodes of the cells upon which the fields can be calculated. When calculating the charge density, the charge must be distributed to those nodes. This is done through weighted charge distribution. The distance between the particle and the 8 nodes of its cell are calculated, and then

the 8 different volumes which cut into the particle are calculated. Those weights determine the amount of charge from the particle distributed to each node.

Once this is done, Poisson's equation can be solved. This is the most computationally intensive part of a PIC code because Poisson's equation is a partial differential equation which is globally coupled. To solve it in the PIC framework, it is discretized across the nodes. Then the a discretized partial differential equation solver can be utilized to calculate the electric potential. For SINATRA, the Finite Difference method is used, as discussed in Section 4.3.1. In order to solve the Finite Difference a linear equation solver must be implemented. SINATRA uses a Gauss-Seidel solver as shown in Section 4.3.2.

Once the electric potential solver is shown to be converged, the electric field can be calculated. In SINATRA the field is calculated using a central difference calculation. This can be seen in Equation 4.8. The equation shown is for the x direction and for a non-boundary node.

$$E_{x,i,j,k} = -\frac{\phi_{i+1,j,k} - \phi_{i-1,j,k}}{2\Delta x} \tag{4.8}$$

E = Electric field

ϕ = Electric potential

x = Distance between nodes in the X direction

i , j and k = The x,y, and z node directions respectively

These three steps only need to be calculated once during the time-step, hence the beauty of the PIC method. During the particle movement part of the DSMC code, these calculations are utilized. Before updating the position of the particle, the velocity is updated. Rearranging and simplifying Equation 4.3 gives the acceleration as $a_x = \frac{q}{m}E_x$. This is used to update the velocity before updating the position of the

particle. Finally, PIC settings are incorporated into the input and output systems of the DSMC and a functioning DSMC-PIC code is born.

4.3 PIC Implementation

Implementing PIC into SINATRA was completed using the algorithm seen in Section 4.2. Two important factors were critical during implementation: robustness and efficiency. SINATRA is built to be able to simulate many different fluid scenarios chosen by the user. Therefore, the PIC portion should also have this generality. Therefore, even though the Finite Difference method requires equal cell sizes, the rest of the algorithms are based on the individual cells so that when the mesh is non-uniform the simulation could be updated with minimum work. It was also implemented in an efficient manner for the DSMC setup and algorithms found in SINATRA. It uses flattened arrays, built-in particle and cell index arrays, and a structure which fits the current parallelization within SINATRA.

4.3.1 Finite Difference

Finite Difference was chosen as the method to discrete Poisson's equation. Finite difference is a simple and common method for discretizing the Poisson equation and is common in PIC codes. There are other algorithms which could be used, but Finite Difference allows for a relatively simple implementation. The derivation below is for a 7-point finite difference for the Poisson equation [28, 29].

The derivation begins with the Poisson equation expanded out to each of the 3 directions.

$$\nabla^2\phi = \frac{\partial^2\phi}{\partial x^2} + \frac{\partial^2\phi}{\partial y^2} + \frac{\partial^2\phi}{\partial z^2} = -\frac{\rho}{\epsilon_0} \tag{4.9}$$

Next the Poisson equation can be discretized in the x , y and z directions as shown in Equations 4.10, 4.11, and 4.12. The indices for x , y and z are given by i , j and k respectively [28].

$$\frac{\partial^2\phi}{\partial x^2}(x_i, y_i, z_i) \approx \frac{1}{h^2}(\phi(x_{i-1}, y_j, z_k) - 2\phi(x_i, y_j, z_k) + \phi(x_{i+1}, y_j, z_k)), \tag{4.10}$$

$$\frac{\partial^2\phi}{\partial x^2}(x_i, y_i, z_i) \approx \frac{1}{h^2}(\phi(x_i, y_{j-1}, z_k) - 2\phi(x_i, y_j, z_k) + \phi(x_i, y_{j+1}, z_k)), \tag{4.11}$$

$$\frac{\partial^2\phi}{\partial x^2}(x_i, y_i, z_i) \approx \frac{1}{h^2}(\phi(x_i, y_j, z_{k-1}) - 2\phi(x_i, y_j, z_k) + \phi(x_i, y_j, z_{k+1})), \tag{4.12}$$

Substituting these into Equation 4.9 gives Equation 4.13 below.

$$\begin{aligned} &\frac{\phi(x_{i-1}, y_j, z_k) - 2\phi(x_i, y_j, z_k) + \phi(x_{i+1}, y_j, z_k)}{h^2} + \\ &\frac{\phi(x_i, y_{j-1}, z_k) - 2\phi(x_i, y_j, z_k) + \phi(x_i, y_{j+1}, z_k)}{h^2} + \\ &\frac{\phi(x_i, y_j, z_{k-1}) - 2\phi(x_i, y_j, z_k) + \phi(x_i, y_j, z_{k+1})}{h^2} = -\frac{\rho_{ijk}}{\epsilon_0} \end{aligned} \tag{4.13}$$

h = Cell side length

This builds a set of linear equations which is a common problem in linear algebra and therefore multiple methods exist to solve it. First, the matrix must be created. This matrix, also called a stencil, only needs to be created once per simulation because it is only dependent on the mesh. Equation 4.14 shows the stencil for a 7-point mesh on a $3 \times 3 \times 3$ node domain.

$$A = \begin{bmatrix} S & I & & I & & & & & \\ I & S & I & & I & & & & \\ & I & S & & & I & & & \\ I & & & S & I & & I & & \\ & I & & I & S & I & & I & \\ & & I & & I & S & & & I \\ & & & I & & & S & I & \\ & & & & I & & I & S & I \\ & & & & & I & & I & S \end{bmatrix} \tag{4.14}$$

where,

$$S = \begin{bmatrix} -6 & 1 & \\ 1 & -6 & 1 \\ & 1 & -6 \end{bmatrix} \text{ and } I = \begin{bmatrix} 1 & & \\ & 1 & \\ & & 1 \end{bmatrix}$$

Boundary conditions in a finite difference model can either be Neumann or Dirichlet. Neumann boundary conditions define the derivative at the boundary while Dirichlet define the value of the solution along the boundary. In this case, that means Neumann boundaries define the electric field while Dirichlet conditions define the value of the electric potential. For PIC simulations, it is best to use Neumann for open or uncharged boundaries while Dirichlet is best for objects like charged surfaces. SINATRA currently only includes Neumann boundary conditions; Dirichlet will be added when the boundary class is updated.[2]Neumann boundaries have been implemented by editing the stencil when the node is along a boundary. At that point Equation 4.15 is used to change the values of the stencil. An average is calculated between

[2]This current condition causes the stencil to be ill posed for a direct solver like Gauss-Seidel [30]. This means that while it will converge upon an answer it is possible that it converged upon an inaccurate result and therefore will skew the results. For current results it has be observed to calculate relatively accurate solutions, however this should be explored in further work and users should be stopped from selecting this situation in the initial conditions.

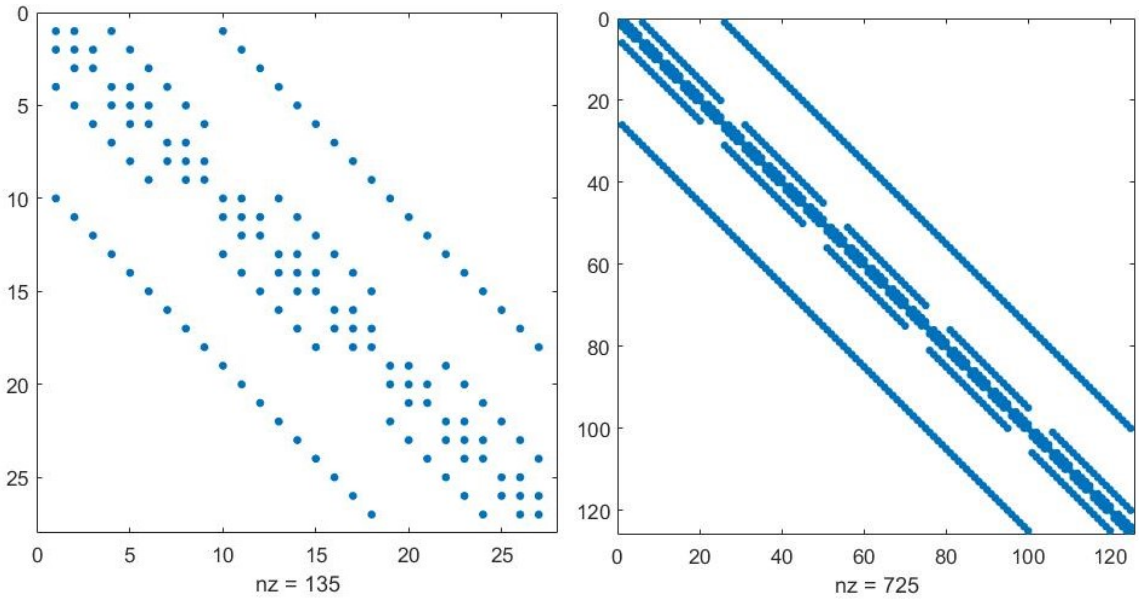

Figure 4.2: SINATRA Sparse Stencil Matrix, shown through the MATLAB® spy command (left) 8 Cells in the mesh (right) 64 cells in the mesh.

the direction of boundary and the perpendicular direction in order to keep the slip boundary along open walls.

$$A_{x_0,y_j,z_k} = \frac{-\phi(x_0, y_j, z_k) + \phi(x_1, y_j, z_k)}{2h} \tag{4.15}$$

This stencil was implemented into SINATRA as a flattened 2D matrix which in itself is two flattened 3D matrices. The electric potential is stored as a flattened 3D matrix. The matrix items are selected through conversion functions that go from 3 dimensions to 1 dimension and vice versa, shown in Equations 4.16 and 4.17. Figure 4.2 shows a MATLAB® *spy* command on SINATRA's stencil. The spy command shows which items in a matrix are non-zero. As seen by these figures the stencil is largely empty. Future work would include using a different data structure and access algorithm to reduce the wasted memory.

$$Index = i \times s^2 + j \times s + k \tag{4.16}$$

$Index$ = The index of the 1D array

i , j and k = The indices of the 3D array

s = The size of one dimension of the 3D array

$$\begin{aligned} i =& \frac{Index}{s^2} \\ j =& \left(\frac{Index}{s}\right) \% \, s \\ k =& Index \% \, s \end{aligned} \tag{4.17}$$

Note: Values rounded down to closest whole number for use as indices

% = The modulus operator which gives the remainder of division

At this stage, the simulation assumes equal cell sizes and refinement across the whole domain. However, the charge density summation is calculated cell by cell, and the electric field is distributed cell by cell. These allow future iterations to have unequal cell sizes.

4.3.2 Gauss-Seidel

The Gauss-Seidel iterative method was chosen to be the linear algebra solver for SINATRA. It was chosen for its simplicity, universality, and robustness. It has one improvement over the simplest iterative method, Jacobi, in that it uses the information as it calculates it. It solves a version of the Equation 4.18.

$$A \cdot x = b \tag{4.18}$$

In terms of SINATRA,

A = Stencil - Matrix of coefficients

x = Electric Potential - The vector to be solved

b = Charge Density - The calculated vector

The Charge Density (b), as defined in Equation 4.6, includes the electron density. In SINATRA the electrons are approximated with a fluid assumption as shown in the Boltzmann relationship (4.7). This thesis will not go into a derivation of the Gauss-Seidel method, but the equation to update the electric potential at each time-step is given by Equation 4.19 [3].

$$x_i^{k+1} = \Big(b_i - \sum_{j=1}^{i-1} A_{ij} x_j^{k+1} - \sum_{j=i+1}^{n} A_{ij} x_j^k\Big)/a_{ii} \tag{4.19}$$

It was discussed previously how to distribute the ion's charge, but not how to combine the ion charge and the electron charge. Therefore, by discretizing Poisson's Equation (4.5), combining it with Equations 4.6 and 4.7 for the charge density, and putting it in matrix form (4.14) we get Equation 4.20.

[3] A derivation of the Gauss-Seidel method can be found in Reference [31]

$$A \cdot \phi = -\frac{e}{\epsilon_0}\Big[n_i - n_o \exp\Big(\frac{\phi - \phi_0}{k\,T_e}\Big)\Big] \tag{4.20}$$

A = The sparse stencil

e = Charge on an electron

ϵ_0 = Permittivity of free space

n = Number density

ϕ = Electric Potential

ϕ_0 = Initial Electric Potential

k = Boltzmann Constant

T_e = Temperature of the Electrons

This shows that the b in Equation 4.18 is the right hand side of the set of linear equations for the Gauss-Seidel solver. Importantly, note that both sides depend on the electric potential. It means that the right hand side must be recalculated at every time-step.

4.4 Results

Two main methods, Solver Validation and Test Cases, have been used to validate the PIC portion of SINATRA. By both testing the individual solver as well as running test cases, SINATRA's new upgrade can be validated as a working PIC code for its methods. All input files used for the test cases can be found in Appendix F. The top of each page of that appendix shows the title of the test case. All simulations can be run using the master branch in the GitHub® version committed on June 14$^{\text{th}}$, 2019.

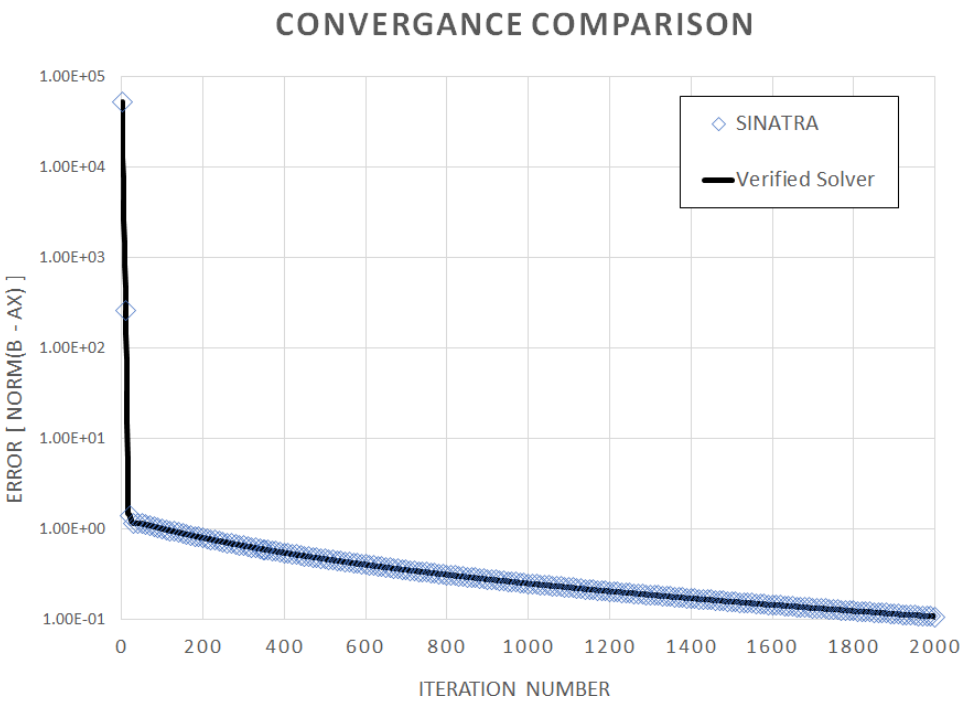

Figure 4.3: Convergence Visualization Error of the Gauss-Seidel solver over 2000 iterations compared to a MATLAB® verified solver. The error is calculated through the vector norm of the difference between the stencil times the electric potential and the charge density.

4.4.1 Solver Validation

SINATRA uses a Gauss-Seidel solver to solve the 7-point 3D stencil. The Gauss-Seidel solver calculates the value of x when given A and b in $A \cdot x = b$. In order to validate the Gauss-Seidel solver, the SINATRA implementation was tested against a validated MATLAB® version[4]. Both solvers were given the same inputs shown in Table 4.1. The important components of the solvers were logged during operation. The results of the two solvers were then compared to confirm that SINATRA's solver is valid. As shown in Appendix E, the solvers were shown to match with only slight rounding errors between them. This is expected in a Gauss-Seidel solver because it has no randomness. Their agreement can also be seen in Figure 4.3.

[4]The validation was done through comparing Wolfram Alpha and the MATLAB® version

Table 4.1: The initial conditions for solver verification

Property	Value	Property	Value
Number of Cells	64	Real to Simulated particles	1×10^{19}
Sphere Model	Hard Sphere	Collision Scheme	Off
Time-step (s)	1×10^{-8}	Total Simulation Time (s)	5×10^{-8}
Number Density	1×10^{24}	Gas Type	Nitrogen
Electron Density	1×10^{12}	Electron Temp (eV)	1
Reference Potential (V)	0	Domain size (m)	$1 \times 1 \times 1$
Stream Temperature (K)	300	X,Y,Z Velocity (m/s)	0,0,0

A visual way to confirm that the solver works is by observing the error as the iterations increase. A few tests were run, and the error of the solver was outputted at each iteration. A 64-cell convergence comparison is shown in Figure 4.3, with an average 24.8 particles per cell. It was compared to the validated solver's error residual and they are shown to match exactly. As seen in the figure, the error drops quickly and then slowly decreases towards zero. This shows numerical stability and convergence. However, it also shows that Gauss-Seidel, while robust, is not optimal for very large simulations. A visualization of the error is a good way to estimate the accuracy of the solution and just as importantly, the soundness of the initial conditions.

4.4.2 Ambipolar Test Case

Two main test cases were performed in order to test the validation of the SINATRA: Ambipolar diffusion and steady state electric flow. Ambipolar diffusion is a typical simulation case for PIC codes. Normal fluids will diffuse out of a domain naturally over time. However, on account of the different speeds between electrons and ions in a plasma, a gradient is set up as the electrons start to move faster than the ions. This gradient causes an electric field which in turn reduces the speed of the electrons while increasing the speed of the ions. This causes a uniform diffusion rate.

Table 4.2: The initial conditions for ambipolar diffusion

Property	Value	Property	Value
Number of Cells	4096	Real to Simulated particles	1×10^9
Sphere Model	Hard Sphere	Collision Scheme	Off
Time-step (s)	1×10^{-7}	Total Simulation Time (s)	5×10^{-5}
Number Density	1×10^{12}	Gas Type	Argon
Electron Density	1×10^{12}	Electron Temp (eV)	100
Reference Potential (V)	0	Domain size (m)	$1 \times 1 \times 1$
Stream Temperature (K)	5000	X,Y,Z Velocity (m/s)	0,0,0

The initial conditions for the ambipolar diffusion were based on diffuse hot plasma consisting of argon gas [11]. They can be seen in Table 4.2. These conditions were chosen by taking the density and temperature of a diffuse hot plasma, then calculating the other parameters according to the requirements for a PIC simulation. The Debye length was calculated to be 0.074 meters. The cell density was selected such that the cell size was approximately the Debye length. The time-step was calculated to be on the order of 1×10^{-7} seconds and the total simulation time was chosen to see significant change in the simulation. The domain was initialized with uniformly distributed particles. The number of particles per cell was 0.85 which should be increased in future simulations for greater accuracy and to avoid numerical heating.

There are two simple ways to confirm that the ambipolar diffusion is working correctly. First is by comparing the density distribution throughout the simulation time. This is done through viewing a slice of the density distribution once every 2×10^{-5} seconds. These can be seen in the full-page Figures 4.4 and 4.5. They are a clear validation of the implementation of the PIC simulation. In the neutral particles, it can be seen that the density near the walls slowly decreases. The sections of high density particles become smaller as more particles leave the domain. However, the charged particles are attracted to each other and join together in the middle to form a high density core of the domain. This slows the diffusion rate significantly.

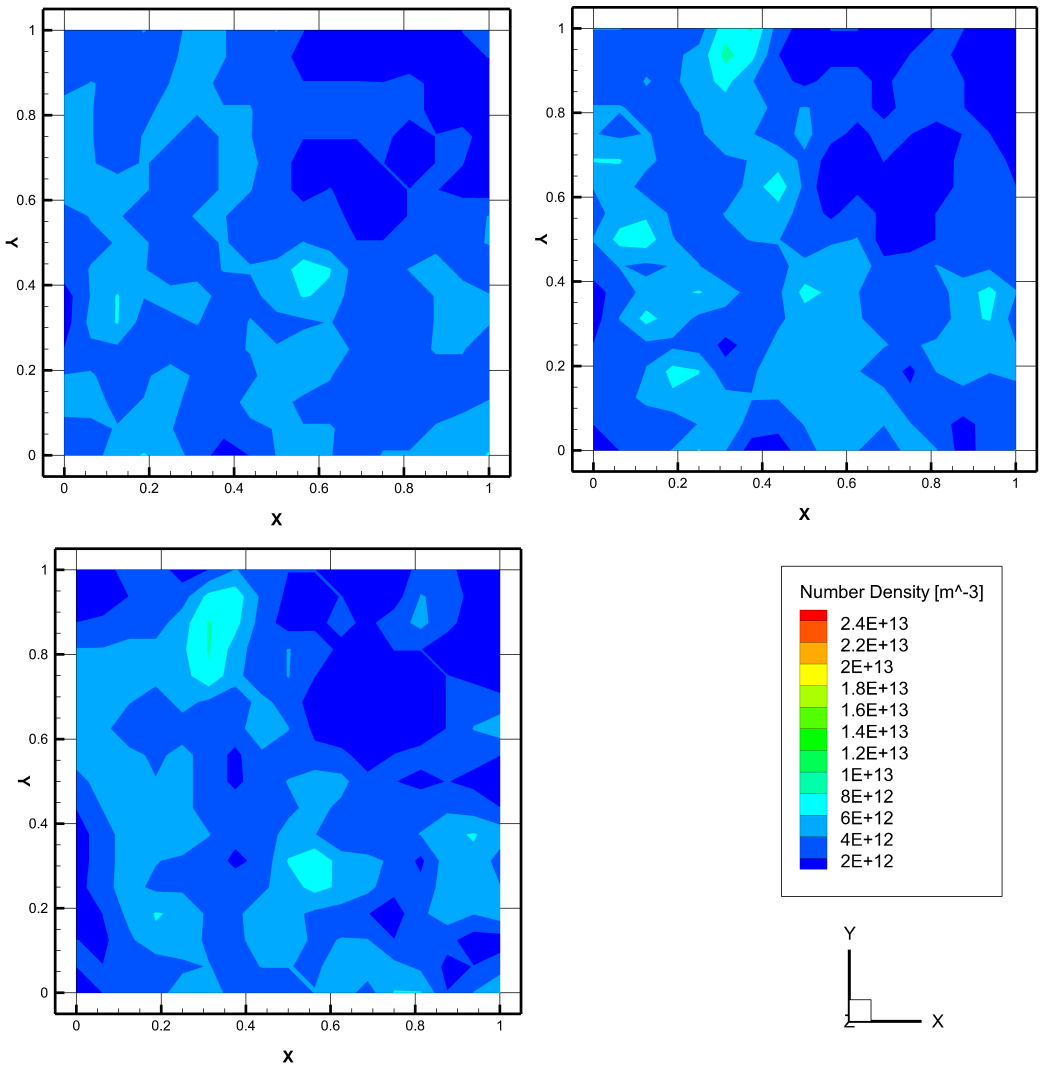

Figure 4.4: Neutral Diffusion Density The domain of neutral diffusion from 1×10^{-5} to 5×10^{-5} seconds. First is top left, then top right, and finally bottom left. This is one slice in the Z direction as indicated by the axis, and the density levels are shown in the bottom right.

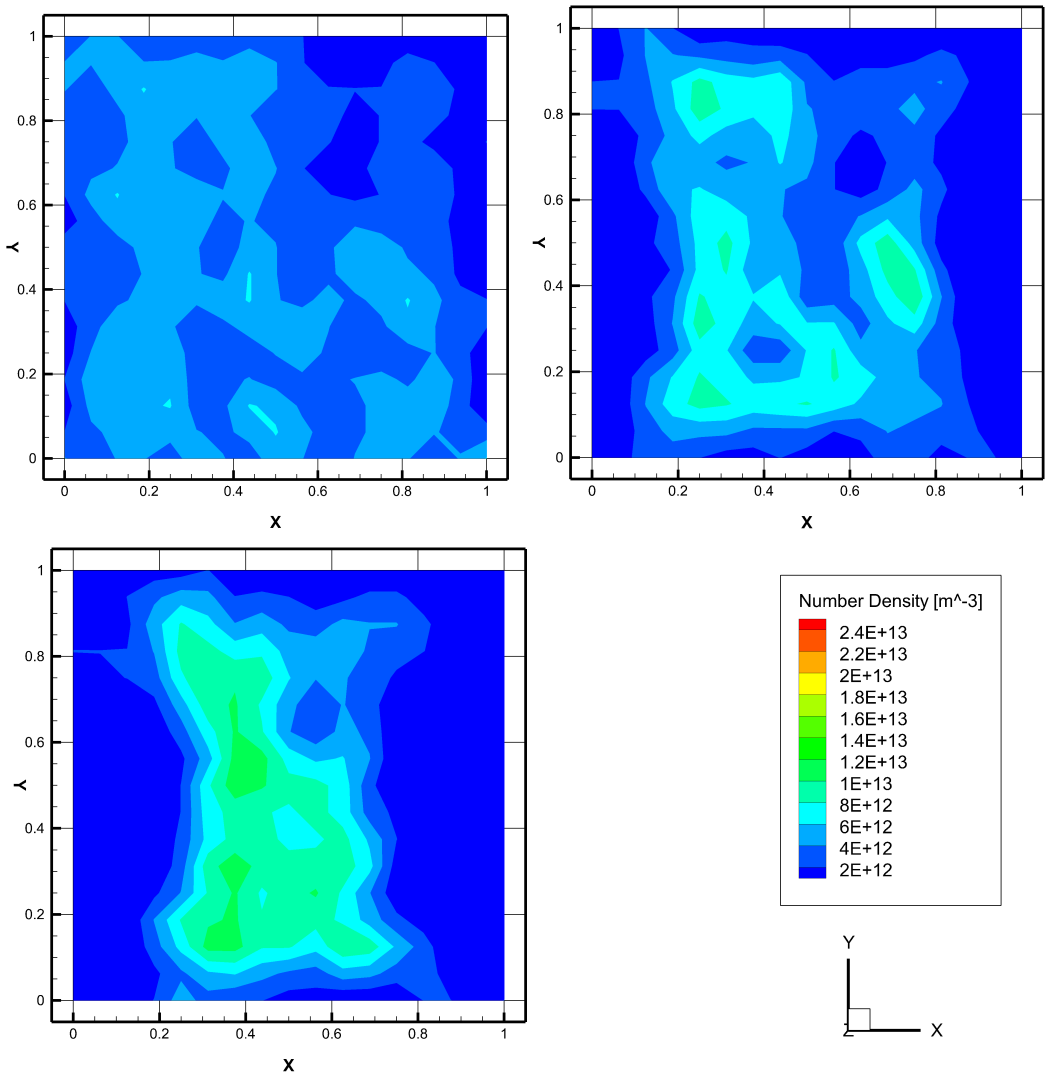

Figure 4.5: Ambipolar Diffusion Density The domain of ambipolar diffusion from 1×10^{-5} to 5×10^{-5}. First is top left, then top right, and finally bottom left. This is one slice in the Z direction as indicated by the axis, and the density levels are shown in the bottom right.

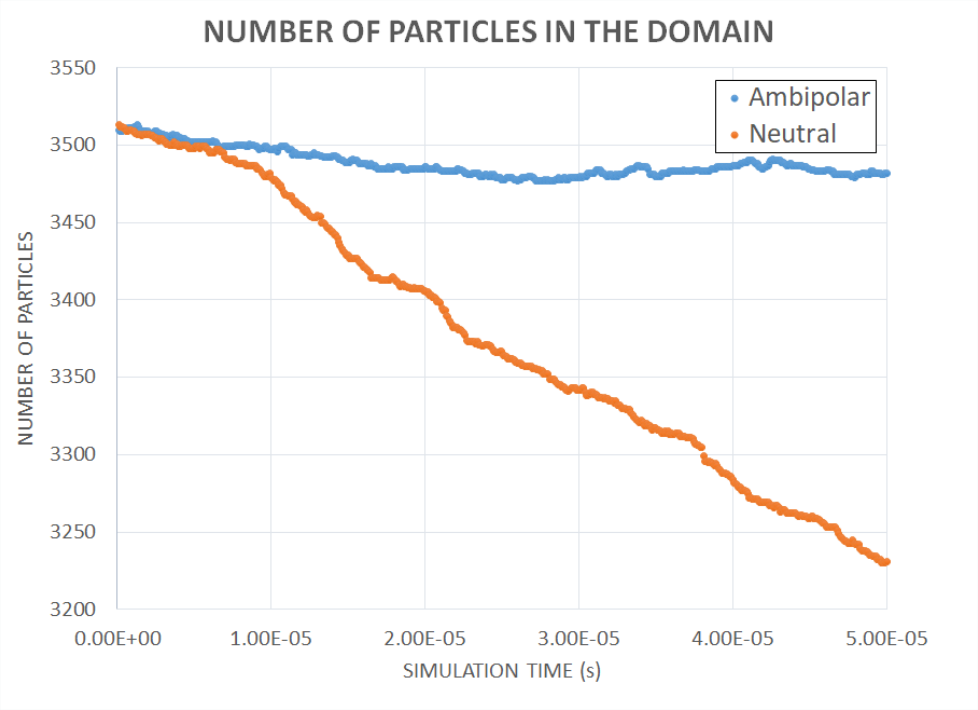

Figure 4.6: Number of Particles in Diffusion The number of particles in the domain for ambipolar diffusion vs neutral diffusion

Another metric to see if the diffusion is working is the number of particles in the domain. The number of particles is shown in Figure 4.6, and it shows the large difference between charged particles and neutrals. Both simulations begin with losing particles at a similar rate. However, within 0.5×10^{-5} seconds they diverge. The diffusion of the neutrals through the boundaries is relatively quick compared to the diffusion of the charge particles. Through the relative difference ambipolar diffusion can be observed.

Finally, an analytical comparison is a strong validation tool for the PIC implementation. For this, the ion flux can be compared to the gradient of the density because they are proportionally related [12]. They are related by the negative ambipolar diffusion coefficient, which is shown in Equation 4.21.

Table 4.3: Ambipolar diffusion coefficient results

X	Y	Z	Flux (Y)	Density Gradient (Y)	RHS of Eqn 4.21 (Y)
4	8	4	0.00E+00	0.00E+00	0.00E+00
4	8	8	0.00E+00	-2.10E+21	0.00E+00
4	8	12	0.00E+00	0.00E+00	0.00E+00
8	8	4	-5.43E-07	2.10E+21	3.08E+22
8	8	8	0.00E+00	0.00E+00	0.00E+00
8	8	12	5.43E-07	0.00E+00	-3.08E+22
12	8	4	-5.44E-07	0.00E+00	3.08E+22
12	8	8	-5.43E-07	0.00E+00	3.07E+22
12	8	12	0.00E+00	0.00E+00	0.00E+00

$$\tau = -D_a \nabla n \tag{4.21}$$

$$D_a = \frac{\mu_i D_e + \mu_e D_i}{\mu_i + \mu_e} \tag{4.22}$$

τ = Ion Flux

n = Density

D_a = Ambipolar diffusion coefficient

μ = Ion and electron mobilities

D = Ion and electron diffusion coefficients

In order to test this equation first the Ambipolar diffusion coefficient is calculated in this simulation to be $5.6627 \times 10^{28}\ m^2/s$ using the code found in Appendix G. Then the gradient in a direction and the flux is calculated over test points within the domain. These were then compared and can be seen in Table 4.3. A few notes on these results are as follows. First, the low number of particles makes the results very discretized and therefore not consistent. Also, the ambipolar diffusion has made some cells have no particles and therefore the densities are actually 0, giving a skewed result. However, what can be seen is the magnitude of the results. This is shown to be within 1 order of magnitude. This is acceptable on account of two reasons.

First, the ambipolar diffusion coefficient assumes a continuum solution and therefore is not completely accurate in this simulation. Secondly, the all neumann boundary simulation means that the simulation may find a solution which is not necessarily the most accurate solution. These two sources of error explain the error in these results.

4.4.3 Steady State Flow

An electric propulsion plume can be approximated by a simulation with all the boundaries as outflows except for a single inlet. That inlet has some initial velocity into the domain. This is the second test case for the PIC implementation into SINATRA. It serves two purposes. First, it is a starting point for a complicated electric propulsion plume. Secondly, it is a strong visual tool to confirm that the poisson equation solver is generating reasonable electric potential values. All boundaries are outflows while the -X boundary is an inlet with velocity going into the domain. Table 4.4 shows the initial conditions for this test case. The domain is uniformly initialized for number of particles per cell (9.85) and given an Maxwellian distribution for velocity of 1000 m/s in the X direction.

Figure 4.7 shows the contours of the potential across a slice of the domain. This figure shows how the electric potential does not have any large numerical instabilities, it does not change shape dramatically between time-steps, and it is relatively regular as expected for an empty domain. To show that as time progresses the simulation is reaching the steady state solution, the maximum difference in the potential between time-steps was calculated and plotted in Figure 4.8. The asymptotic nature of the difference approaching zero shows that this simulation is converging upon a steady state solution to the given initial conditions. This is not the Gauss-Seidel solver slowly converging upon a solution because the 5% solver requirement was used which ensures that the Gauss-Seidel is steady before moving on to the next time iteration. This test

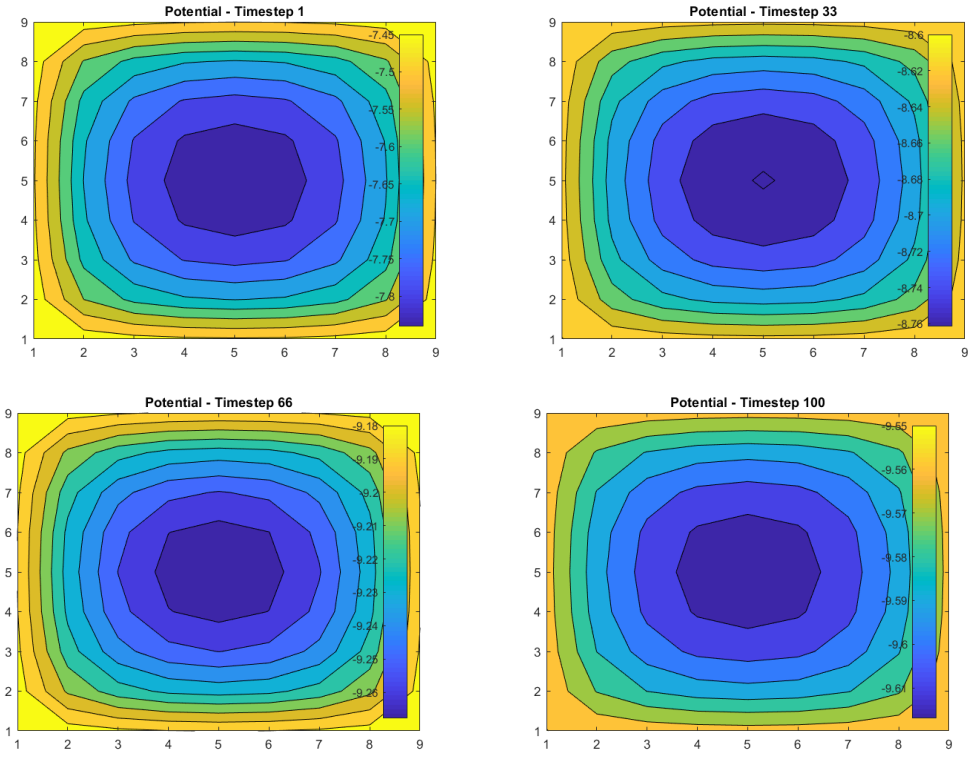

Figure 4.7: Steady State Potential The domain of steady state flow from 1×10^{-8} to 1×10^{-6}. First is top left, then top right, and then bottom left, and finally bottom right. This is one slice in the Z direction at node 4. The color bar shows the value of the potential.

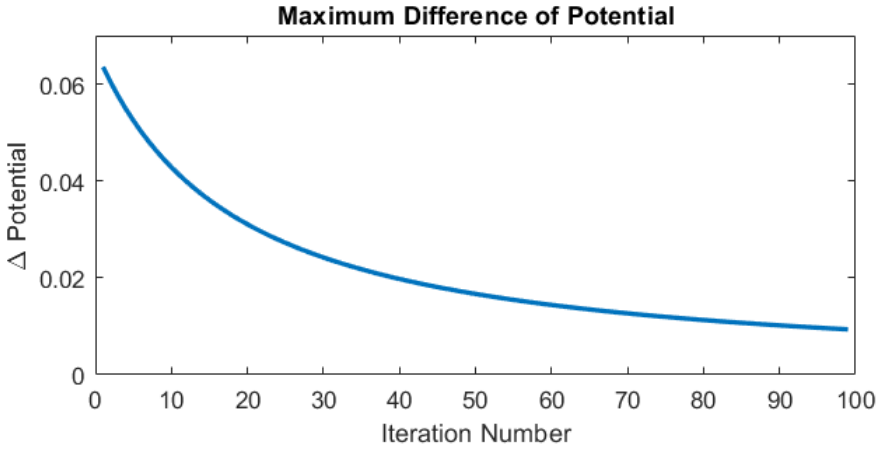

Figure 4.8: Maximum Difference in Potential The maximum difference in potential across one slice of the domain in the Z direction at node 4. The steady state convergence can be seen in the approach to 0 difference in potential.

Table 4.4: The initial conditions for steady state flow

Property	Value	Property	Value
Number of Cells	512	Real to Simulated particles	1×10^{9}
Sphere Model	Hard Sphere	Collision Scheme	Off
Time-step (s)	1×10^{-8}	Total Simulation Time (s)	1×10^{-6}
Number Density	5×10^{12}	Gas Type	Argon
Electron Density	1×10^{12}	Electron Temp (eV)	1
Reference Potential (V)	0	Domain size (m)	$1 \times 1 \times 1$
Stream Temperature (K)	5000	X,Y,Z Velocity (m/s)	1000,0,0

case validates a correct implementation of charged particles as well as well-suited initial conditions to study complicated electric propulsion plumes.

4.4.4 Execution Time Study

In order to examine the performance of SINATRA after the inclusion of PIC, an execution time study was performed. The convergence condition was tested as both the absolute error being less than 0.05 Volts and a less than a 5% difference in the error from the last 50 iterations. Simulations were run keeping the same number of particles but changing the mesh size, as well as holding the mesh size and changing the number of particles. It is also compared to a similar non-charged simulation. The results can be found in Table 4.5.

These simulations used the same settings as ambipolar diffusion. There were 2500 maximum iterations and it was run for 500 time-steps. First, the large difference (over 98%) between the neutral simulation and similar charged particle simulations is evident. This is on account of the large number of iterations necessary to solve the electric potential. Secondly, the difference between the 5% percent change convergence condition and the 0.05 Volts absolute error condition is significant, on the order of 65%. This can be attributed to the fact that if a time-step does not converge, the next time-step starts where the last one ended, because electric potential is

Table 4.5: PIC Execution Time

	First Iteration	Time-step Average	Total Time
Neutral	2 sec	< 1 sec	3 min, 4 sec
5% error convergence	1 min, 56 sec	24 sec	199 min, 44 sec
0.05 V error convergence	18 min, 46 sec	18 min, 46 sec	578 min, 11 sec[6]
64 Cells, 5% error	< 1 sec	< 1 sec	15 sec
10 times more particles	1 sec	< 1 sec	26 sec[7]

not reset between time-steps. Consequentially, the solver keeps trying to reduce the error throughout all the time-steps. In the percent change simulation, it can be seen that the first simulation reaches convergence, and the sequential time-steps only need to solve the slight difference in the charge density caused by the moving particles. However, the first iteration of the absolute error simulation takes the same amount of time as the other iterations and therefore it is continually reaching the maximum time-steps attempting to reduce the error[5].

Through the results in Table 4.5, it can be seen that the execution time is more heavily influenced by the number of mesh cells than the number of particles. When the number of cells is reduced to 64, the execution time is reduced from over 3 hours to 15 seconds. It follows that the combination of an octree mesh and a non-optimal code dramatically changes the simulation time. The final simulation uses a 64-cell mesh and increases the number of simulated particles by 10 times. Even with the increased number of particles there is only an 11-second difference. This shows how the number of cells has a larger influence than the number of particles. It also shows that parallization makes a negligible difference in the PIC code unless the number

[5]This was confirmed by an examination of the output logs

[6]Extrapolated from first 25 time-steps, assuming same average up to time-step 100, then 24 seconds per time-step.

[7]Extrapolated from 30 time-steps, errors on account of too many particles in each cell

of simulated molecules is extremely large. It is future work to find a more efficient solver in order to increase the simulation accuracy of SINATRA.

Chapter 5

CONCLUSIONS

The goal of this thesis is three-fold. The first goal is to help develop a homegrown DSMC simulation tool for Cal Poly. The second goal is to implement charged particles into the DSMC to create a hybrid DSMC-PIC simulation. The third goal is to design the code architecture and systems to have both a low probability of becoming unusable and a similarity to the upcoming homegrown fluid simulation. These are all intermediary goals which work towards the final end of creating a full simulation of an electric engine. This thesis shows that the three goals have been accomplished through the latest upgrade of the new homegrown code, SINATRA.

Building a homegrown CFD code is a complex task. The early developers have worked together to build a robust and flexible DSMC code base. This has been demonstrated through Alliston and Galvez's theses [18, 19]. It has the ability to model many different flow conditions, species, and boundary conditions. Its execution time (i.e. under 2 minutes for 2 million particles) allows for accurate simulations in an institution research setting. It has been validated across its various collision and sphere models.

In order to eventually model a full electric thruster, this part of the simulation must be able to model charged particles for the plume. This has been achieved by making SINATRA a DSMC-PIC hybrid. A simple and robust method is used for discretizing Poisson's Equation: The Finite Difference method. This involves creating a 7-point 3D stencil, Neumann boundary conditions, slip conditions through averaging. For solving the resulting set of linear equations, the Gauss-Seidel iterative method

was chosen. The implementation in SINATRA was verified against a verified solver and shown to converge on the 3D stencil. The accuracy of the PIC implementation has been validated through two important test cases, ambipolar diffusion and steady state flow. In the ambipolar diffusion test case, it is clear that the charged particles create a force which greatly reduces diffusion so that within 5×10^{-5} seconds only 0.071% of the particles leave the domain. With the steady state flow, the potential was seen to quickly assume a steady state solution and change slowly as the particles moved through the domain.

The code base has been converted from a small developer's program to a university standard code. It is managed through GitHub® in order to be used by multiple university developers. It has been upgraded with a user manual, consistent and simple documentation, and a simplified workflow. Two large bottlenecks, particle pushing and particle linking, have been greatly mitigated through parallization and increased data throughput so that timely simulations may be made. In addition, a distribution method has been created with easy-to-use executables and a GUI for user control.

There are still many ways to improve upon SINATRA. Those are discussed in Section 5.1. However, SINATRA is a strong first iteration of a university level DSMC-PIC code base. This work will help SINATRA reach its goals and Cal Poly will enabled to do novel research in modeling full electric thrusters.

5.1 Future Work

Because of the nature of SINATRA, it is expected for there to be a large amount of future work. A critical part of SINATRA is that it will be developed into a state-

of-the-art homegrown code which Cal Poly can use to help develop new aerospace technology.

5.1.1 Boundaries

One part of SINATRA which is underdeveloped is the handling of boundaries. Currently, SINATRA handles the main 6 boundaries of a cube. It has the capability to define the type of wall and the characteristics of the particles flowing through that wall. However, it ignores any boundaries inside of the domain, cannot break a boundary into multiple sections. While dealing with boundaries inside the domain will not be needed for a plasma plume, it will be important to be able to split the boundaries so that part of the wall can simulate the thruster nozzle. The path forward for boundaries is twofold. First, a boundary class needs to be built within SINATRA which allows the user to specify sections which will be different from the rest of the domain. At that stage it would be reasonable to simulate a thruster nozzle.

The next stage would be to build Cart3D^{TM} integration. Cart3D^{TM} is a meshing software that can create an octree mesh which includes internal domain boundaries. This allows a user to import a Computer-aided design (CAD) file and get out a mesh which SINATRA can understand. This is critical for upper atmosphere calculations around aircraft or spacecraft bodies. It can also be used for objects in Low Earth Orbit. The Cart3D^{TM} tool will allow users to specify what each external and internal boundary type is and is able to split these boundaries into much smaller pieces. Cart3D^{TM} can also dynamically change the mesh size depending on distance to a wall and other user specifications which will allow for more accurate DSMC-PIC simulation data. This integration will need to be completed before SINATRA can create useful simulation results. The future researchers can also choose a difference meshing software depending on future requirements.

5.1.2 Electric Thruster

As mentioned above, in order for an accurate electric thruster simulation, there will need to be changes in how boundaries are handled in SINATRA. Updating the boundaries is the most important change that will be needed for accurate simulations of electric thruster plumes. The Poisson equation solver would need to be upgraded as well. Currently, a finite volume solver is being used. This is a robust solver which is well-researched and understood, but it has a few restrictions. First, and most importantly, the solver expects the mesh to be evenly sized across the entire domain. This works with the current version of SINATRA's home-built meshing software; however, once the mesh is not completely uniform, the Poisson solver will break. This solver can only handle straight boundaries, which is acceptable because the DSMC portion uses straight approximations of curves through the octree mesh. There are many other options for a Poisson solver that have been explored in PIC research. For example, the conjugate gradient solver might be a good option for SINATRA. Conjugate difference requires a symmetric positive definite matrix which the Finite Difference method creates [28]. This upgrade would also greatly reduce the execution time, and therefore allow for larger and more accurate simulations.

5.1.3 Charged Particles

There are many facets to simulating charged particles. While the author has captured the largest features, there are many other physical attributes which make charged particles a complicated and interesting subject. There are two main physical properties which would be the most likely candidates to be added to SINATRA. They are charged collisions and surface interactions.

When charged particles are involved, there are many new types of particle collisions. Some types are ionization and recombination collisions. These are being ignored in SINATRA because the electrons are being modeled as a fluid. However, if magnetic fields were to be included, for example for a magnetic nozzle, or if the grids of an ion thruster need to be modeled, then the fluid assumption would no longer be valid. The simulation would have to drastically reduce its simulation time-step for the fast electrons and also need to consider ionization and recombination collisions. There are also chemical interactions that are also not being considered in the scope of this project. Both of those types of collisions will most likely not be needed for accurate plasma plume simulations. In order for these collisions to be included, the collision class will need to be updated to be able to handle multiple schemes within the same simulation.

However, charge exchange collisions will need to be included eventually. Charge exchange collisions are instances where the electron orbital cloud of an ion and a neutral particle will interact [32]. There is a possibility in these collision for an electron to be stripped from the neutral atom and become attached to the charged ion. The charge is exchanged between the two particles; the neutral atom becomes a charged ion and the charged ion becomes a neutral atom. However, there is no significant change in momentum. This type of collision is common and significant in electric thruster plumes. While most thrusters are efficient at ionizing the propellant, there are still neutral atoms which come out of the chamber and into the plume. Their relatively low velocities cause the relative density of the neutral atoms to be high near the thruster nozzle. Charge exchange collisions (CEX) are therefore likely. The resulting slow-moving ions are very susceptible to the radial component of the electric field set up by the plume. While the fast-moving ions will diverge, they do so at a low angle because of their high initial velocity. However, these new ions are moving slowly and therefore are more easily affected by the radial electric field. They create what

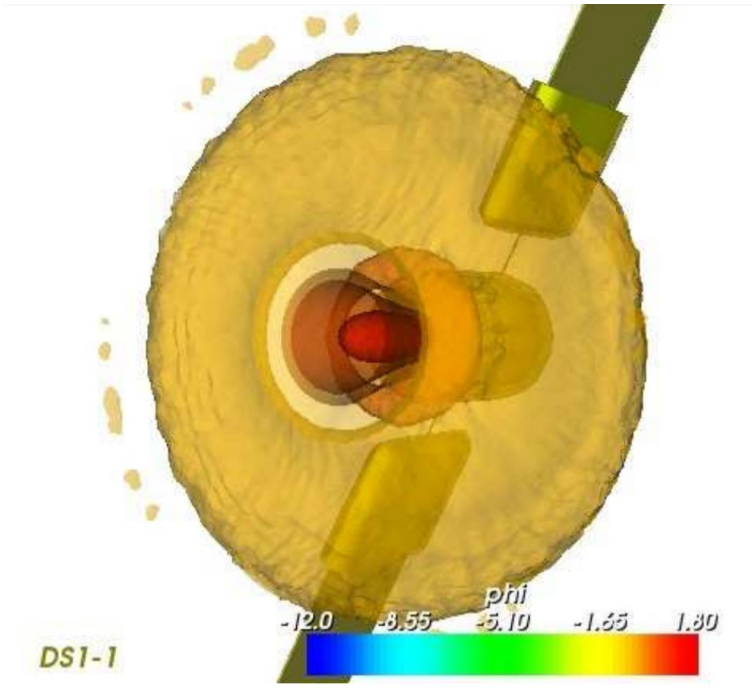

Figure 5.1: Visualization of CEX wings [33]

is called CEX wings, seen in Figure 5.1 [33]. This radial direction of charged ions can cause major interactions with the spacecraft including contamination, sputtering, and spacecraft charging. Therefore CEX interactions should be included when doing full spacecraft analysis.

Another important quality of charged particles is their surface interactions. When high energy particles collide with metal surfaces they sputter off material and degregate the surface. This is an important design consideration in spacecraft with electric propulsion. SINATRA models various types of surface interactions through its boundary handling. There can be inflow, outflow, specular and diffuse walls. However, especially with CEX collisions, charged particles can interact with a spacecraft surface and impart their charge onto it [34]. The surface can be modeled as a grounded conductor or as a perfect insulator or as a mix between the two. This type of modeling will need to be included into SINATRA whenever boundaries are introduced into the domain.

5.1.4 SINATRA Efficiency and Capability

There are many sections in SINATRA that need to be updated before SINATRA can be used as a cutting edge research tool. It was developed by mechanical and aerospace engineers, not by computer scientists. Therefore, many of the algorithms, storage methods, and memory access are not optimized. The author has removed the largest and simplest bottlenecks, but there are other areas that need to be optimized. The next largest bottleneck is the sampling of data. When SINATRA samples data from the simulation, it prints out text files. This printing is one of the slowest portions of the simulation. Tecplot^{TM} has an API which allows users to print binary files and Tecplot is able to input those binary files. This was attempted by the original developers, but they were not successful. It requires a deeper understanding of C++ and binary. There are many examples of possible optimization throughout the code which can be fixed by a developer with that type of skill set. It may also be possible to switch to the Visualization Toolkit VTK which has other features which may make sampling data much simpler and quicker.

Within a computer scientist skill set lies parallelization. Similar to the bottlenecks above the author has implemented a simple version of parallelization; however, there are many better ways to implement it. It could be implemented by splitting the domain into multiple pieces, which is a possibility on account of the regularity of the octree mesh. However, this may not be the best option because particles must be transferred between the domains which is very computationally expensive. Another option would be to split a larger section of each time-step into many parts. The simulation could also be parallelized by having each core run the same simulation. Then the multiple different simulations could be averaged as a way to reduce the randomness of DSMC so that an accurate solution is calculated. Parallelization tech-

niques are still on the cutting edge of computer science, and therefore this would be a fruitful project for a developer with experience in this area.

Within the DSMC community, there are many schools of thought about the best way to work with time-steps and mesh sizes [8]. It is possible to have variable time-steps as well as variable mesh sizes. It is also possible for the time-step to change for each particle depending on their velocity and the size of the mesh cell they are within. There are algorithms which create a mesh which changes cell size depending on the average number of particles in a cell. This creates a mesh which changes with the flow and eventually sets up an optimal mesh for that steady state flow condition.

SINATRA is currently at its first iteration; therefore it uses a fixed time-step for all particles and a fixed mesh. Upgrading SINATRA to a more complicated time-step and mesh algorithm would be a good project for a future developer. In order to keep charged particle capability, the Poisson solver would have to be upgraded at the same time because it is currently based on the fixed mesh. This upgrade could greatly reduce SINATRA's execution time; therefore, it would allow higher resolution and accurate simulations. It would be beneficial to implement a user choice for different Poisson solvers so that a user can compare various solvers and use their various advantages depending on their simulation requirements. It would also be useful to include an option for the user to have SINATRA assume a linearized Boltzmann equation, which assumes that $n_i - n_e << n_i$. This would allow a much quicker calculation of the potential and be a good low accuracy solver for the beginning stages of complicated simulations.

The stencil implementation will also need to be upgraded. When non-uniform Cartesian grids are included in SINATRA the stencil implementation will fail. The Finite Difference Method requires an uniform mesh. However, the Finite Volume method may be a good alternative to be used for the new mesh types. It will need to be

upgraded to be much more robust to accommodate complicated meshes. The stencil implementation can also be upgraded. Currently the sparse matrix wastes a large amount of computing power through having a large empty matrix and multiplying many zeros. This can be improved thorough either sparse matrix calculations or by calculating the required row of the matrix only when going to do operations upon it. Finally, the future developers will need to be careful when using a non-uniform Cartesian grid and depositing charge on account of the octree structure.

Another area of possible improvement is within the octree search algorithm. Future developers will need to start with confirming that the current search algorithm fits the expected $nlog(n)$ order where n is the number of items in the search. Once confirmed other advanced algorithms can be implemented including aligning the octree mesh to an integer system so that the integer positions can be used as inputs into the data matrix within the calculations.

5.1.5 Systems Operations

It will be important to continually update the systems engineering sections of SINATRA. The author has set up systems which will hopefully be helpful at keeping SINATRA up-to-date and relevant, but they will need to be monitored and maintained

First, the simple distributions need to be kept up-to-date. There are distributions for Windows, Linux, and Mac. If SINATRA continues to grow at Cal Poly, an official release website with version control can be set up. Until then it will be released through .zip files being sent to the new user; therefore, the distributions need to be up-to-date with the current stable version of SINATRA, input files, and the GUI.

The GUI will also need to be updated as there are changes to the input class. Whenever the input class requires a different way to input variables to the simulation, the GUI will also need to be changed to accommodate for those changes. It will also have to be updated when the boundary class or output class changes their user interface. It is not difficult to keep the GUI up-to-date, but it can easily become obsolete if it is left alone during further development.

The author has ensured that the GitHub® repository is kept clean and up-to-date as much as possible. It will be necessary to use the GitHub® repository while developing new code. One pitfall could be that new developers only develop on their local machines and ignore the GitHub® repository. This habit could ruin the continuity of the version control of GitHub® and more importantly could make it harder for new developers to add their contributions. The GitHub® should be kept as a version of the code which can be easily shared with new developers, so they are not be confused nor distracted by extra files and information. If SINATRA is well taken care of, it will become a legacy code that will allow Cal Poly to shine as a school with advanced modeling skills and allow new and revolutionary technology to come from "Learning by Doing".

BIBLIOGRAPHY

[1] "xkcd: Work." `https://xkcd.com/1741/`, 2019.

[2] A. J. Chorin, "Numerical Solution of the Navier-Stokes Equations*," tech. rep., New York University, 1968.

[3] A. L. Garcia and F. Baras, "Direct Simulation Monte Carlo: Novel Applications and New Extensions," tech. rep., San Jose State University, Universite Libre de Bruxelles, 1997.

[4] S. Nguyen-Kuok, *Theory of Low-Temperature Plasma Physics*. Springer International Publishing Switzerland, 2017.

[5] P. T. Gressman and R. M. Strain, "GLOBAL CLASSICAL SOLUTIONS OF THE BOLTZMANN EQUATION WITHOUT ANGULAR CUT-OFF," *Source: Journal of the American Mathematical Society*, vol. 24, no. 3, pp. 771–847, 2011.

[6] M. A. Gallis, J. R. Torczynski, S. J. Plimpton, D. J. Rader, and T. Koehler, "Direct Simulation Monte Carlo: The Quest for Speed," tech. rep., Sandia National Laboratories, 2014.

[7] C. C. Dartmouth College, "BASIC," tech. rep., Dartmouth College, 1954.

[8] G. A. Bird, *The DSMC Method*. The University of Sydney, 1.2 ed., 2013.

[9] A. Morris, *Simulation of rocket plume impingement and dust dispersal on the lunar surface*. Doctor of philosophy, The University of Texas at Austin, 2012.

[10] G. J. LeBeau and F. E. Lumpkin, "Application highlights of the DSMC analysis code (DAC) software for simulating rarefied flows," Tech. Rep. 6-7, Lyndon B. Johnson Space Center, 2001.

[11] Naval Research Laboratory, "NRL Plasma Formulary 2018," tech. rep., Naval Research Laboratory, 2018.

[12] D. M. Goebel and I. Katz, "Fundamentals of Electric Propulsion: Ion and Hall Thrusters JPL SPACE SCIENCE AND TECHNOLOGY SERIES," tech. rep., Jet Propulsion Laboratory, California Institute of Technology, 2008.

[13] O. Räisänen, "Electrostatic Ion Thruster." `https://commons.wikimedia.org/w/index.php?curid=20308289`, 2019.

[14] S. Mahalingam, *PARTICLE BASED PLASMA SIMULATION FOR AN ION ENGINE DISCHARGE CHAMBER.* Doctor of philosophy, Wright State University, 2007.

[15] J. Wang, "Ion Thruster Modeling: Particle Simulations and Experimental Validations," Tech. Rep. August 2014, Virginia Polytechnic Institute and State University, 2003.

[16] J. M. Fife and M. R. Gibbons, "The Development of a Flexible, Usable Plasma Interaction Modeling System," *AIAA*, no. 4267, 2002.

[17] M. Celik, M. Santi, S. Cheng, M. Martinez-Sanchez, and J. Peraire, "Hybrid-PIC Simulation of a Hall Thruster Plume on an Unstructured Grid with DSMC Collisions," tech. rep., Massachusetts Institute of Technology, 2004.

[18] D. Galvez, *the Development and Validation of Sinatra: a Three-Dimensional Direct Simulation Monte Carlo (Dsmc) Code Written in Object-Oriented*

C++ and Performed on Cartesian Grids. Master's thesis, California Polytechnic University, San Luis Obispo, 2018.

[19] R. Alliston, *An Examination of Different Collision Schemes for SINATRA: A Three Dimensional, Object Oriented Direct Simulation Monte Carlo Program*. Master of science, California Polytechnic State University, San Luis Obispo, In Progress.

[20] A. Gay, *Fluidic Plasma Simulator*. PhD thesis, California Polytechnic State University, San Luis Obispo, In Progress.

[21] G. A. Bird, *Molecular Gas Dynamics*. Claredon Press, 1976.

[22] "How to explain object-oriented programming concepts to a 6-year-old." `https://medium.freecodecamp.org/object-oriented-programming-concepts-21bb035f7260`, 2019.

[23] WhiteTimberwolf, "Octree2." `https://en.wikipedia.org/wiki/Octree{#}/media/File:Octree2.svg`, 2019.

[24] "github-flow.png (583276)." `http://files.programster.org/tutorials/git/flows/github-flow.png`, 2019.

[25] "HPC Concepts - Cal Poly, San Luis Obispo." `https://aero.calpoly.edu/technology/high-performance-computing/bishop-hpc-cluster/hpc-concepts/`, 2019.

[26] M. A. Lieberman and A. J. Lichtenberg, *PRINCIPLES OF PLASMA DISCHARGES AND MATERIALS PROCESSING Second Edition*. Hoboken, New Jersey: John Wiley & Sons, Inc., 2005.

[27] "The Electrostatic Particle In Cell (ES-PIC) Method." https://www.particleincell.com/2010/es-pic-method/, 2019.

[28] D. Stonko, S. Khuvis, and M. K. Gobbert, "Numerical Methods to Solve 2-D and 3-D Elliptic Partial Differential Equations Using Matlab on the Cluster maya," tech. rep., University of Maryland, Baltimore County, 2014.

[29] "The Finite Difference Method." https://www.particleincell.com/2011/finite-difference-method/, 2019.

[30] W. L. Briggs, *Numerical Recipes*. Cambridge University Press, 2nd ed., 1987.

[31] G. F. Golub and C. F. Van Loan, *Matrix Computations*. The Johns Hopkins University Press, 1996.

[32] E. Board and R. Beig, *Lecture Notes in Physics*. Springer-Verlag Berlin Heidelberg, 2008.

[33] L. Brieda, J. Pierru, R. Stillwater, and J. Wang, "AIAA 2003-5020 Development of A Virtual Testing Environment for Electric Propulsion," tech. rep., Virginia Polytechnic Institute and State University, Blacksburg, 2003.

[34] A. Barrie, *Modeling Differential Charging of Composite Spacecraft Bodies Using the Coliseum Framework*. Master of science, Virginia Polytechnic Institute and State University, 2006.

[35] I. Boyd, "Simulation Of Electric Propulsion Thrusters," Tech. Rep. 0704, University of Michigan, 2011.

[36] G. A. Bird, *Molecular Gas Dynamics and the Direct Simulation of Gas Flows*. New York: Oxford University Press Inc, 1994.

[37] A. D. Greig, *Pocket Rocket: An electrothermal plasma micro-thruster*. Doctor of philosophy, The Australian National University, 2015.

APPENDICES

Appendix A

EXAMPLE BATCH SCRIPT

Seen below is an example batch script written for Windows™. It can be found in the resources directory as SampleWorkflow.bat.

```
REM This is a sample workflow which simulates a empty domain
REM It meshes a 64 empty domain
REM Then it runs a simple empty domain simulation
REM Finally, it generates a gif of the particles in the domain


REM Run the input program
cd ..\inputProgram
Mesh.exe MeshInput.txt

REM Run the Simulation program
cd ..\simulationProgram
DSMC.exe ..\resources\Complex_Input.txt

REM Run the output program
cd ..\outputProgram
matlab -r FlowVelocityDistributionAnim()
```

Appendix B

DOXYGEN CLASS AND FILE LISTS

Classes

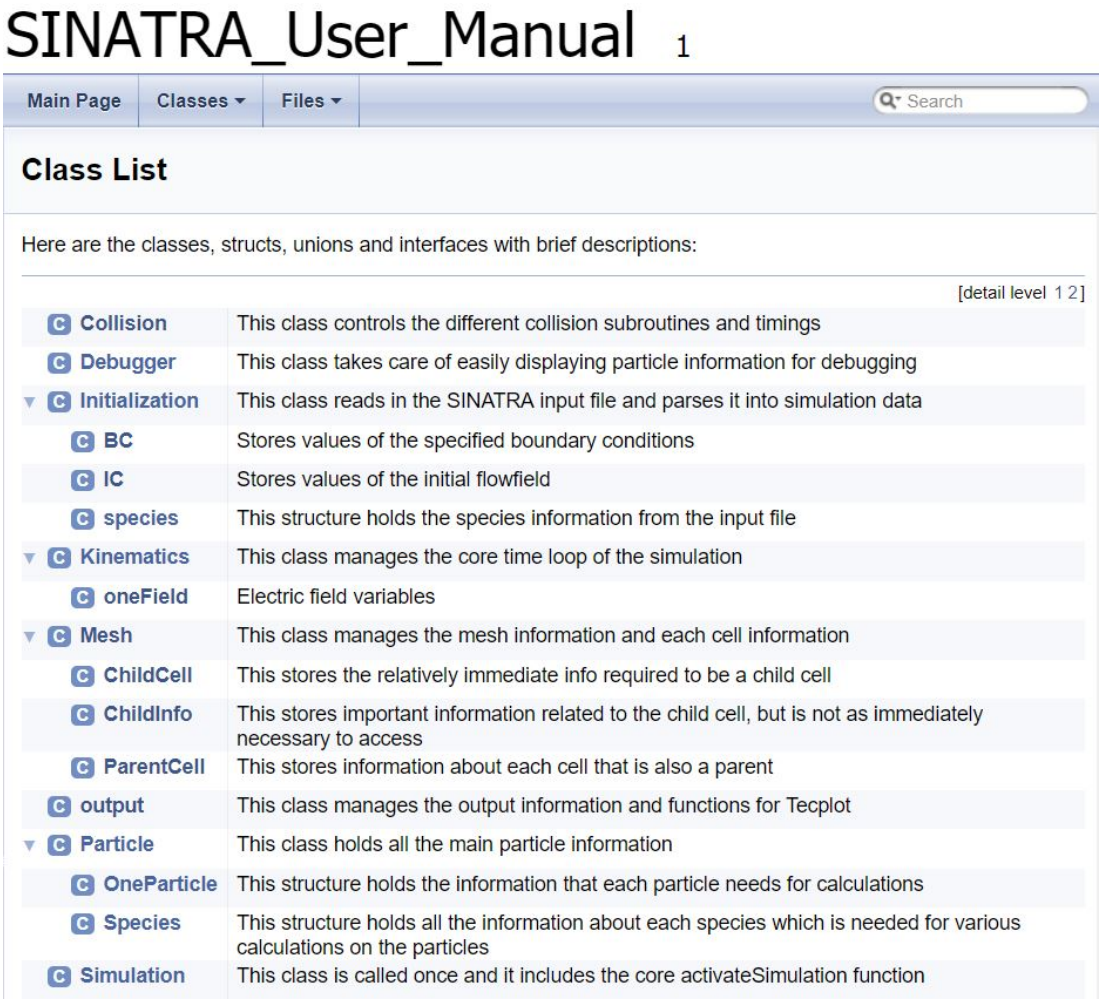

Figure B.1: List of all Classes commented through Doxygen

Files

SINATRA_User_Manual 1

Main Page | Classes | Files | Search

File List

Here is a list of all files with brief descriptions:

[detail level 1 2]

▼ inputProgram	
FEMeshGenerator.cpp	A simple octree mesh generator
FEMeshGenerator.h	A simple octree mesh generator header file
mainInput.cpp	This file runs the input program on an empty domain
▼ simulationProgram	
Collision.cpp	This contains the functions for the **Collision** class
Collision.h	This header file defines the **Collision** class
Debugger.cpp	This file defines the functions used in the **Debugger** class
Debugger.h	This file sets up the class which helps with debugging
input.cpp	This file contains the functions which handle the input of text files
input.h	This file sets up the class for parsing the input file
Kinematics.cpp	File which defines the **Kinematics** class and functions
Kinematics.h	Include file for the **Kinematics** class
main.cpp	**Main.cpp** is the controlling file. It starts the simulation and times it
Mesh.cpp	This file defines the functions which store the mesh information
Mesh.h	This file sets up the information about the mesh
output.cpp	This file handles the output functions for Teccplot and the matlab scripts
output.h	This file sets up the output class for printing information
particle.cpp	This declares the **Particle** class and the functions which mostly depend on just particle calculations
Particle.h	This header file defines the public and private attributes of the **Particle** Class
Simulation.cpp	This is the main simulation file which controls SINATRA
Simulation.h	Defines the simulation class which controls the simulation flow
TecplotWrite.cpp	Code used to create a data file that Tecplot will accept
TecplotWrite.h	This file is the header for functions which write Tecplot files

Figure B.2: List of all Files commented through Doxygen

Appendix C

SIMPLE DISTRIBUTION WALKTHROUGH

Seen below is a simple tutorial for starting with SINATRA. It can be found in the documentation/SimpleDistribution directory as ReadMe.md within each specific OS's directory.

```
SINATRA Simple Distribution Manual
Dominic Lunde


Hello! Welcome to a tutorial on how to use SINATRA.

First, requirements
1) Windows OS
2) Matlab or Matlab runtime - for output only

This will walk you through a simple simulation case.
Then it will show you how to use the GUI and then you can have fun!



Step 1 - Mesh

We must create a mesh file. This is done through
 the Mesh.exe executable.
The MeshInput.txt file is already set up to create
 8 cells through 1 cut.

Therefore we can run the mesher by either:
a) Dragging MeshInput.txt onto Mesh.exe
b) Running "Mesh MeshInput.txt" into a
 command window set to this directory

After this is done, you should see your
 mesh file as a .plt file appear. Good!
```

```
Step 2 - Simulate

Now we can simulate some particles.
Complex_Input.txt is set up to use
 8 cells and run 10 timesteps

Therefore we can run the mesher by either:
a) Dragging Complex_Input.txt onto DSMC.exe
b) Running "DSMC Complex_Input.txt"
 into a command window

All done! You will see a file called SINATRA_OUTPUT.txt.
This shows you very simple time step info.
You will also see SINATRA_uniform_properties_000001.plt.
This has Tecplot data from the first time step.


Step 3 - Analysis

We want to see those particles move right?
We need to change some things first.
Open Complex_Input.txt and go
 to the "Output Information" Section
Between the Tecplot Sample Frequency and the *** add this.

Particle Animation
0
1
VelocityOutput\velocityFromParticles_

The first line is the trigger word, next is the start time,
 then the frequency of output, and finally the base filename.
--Create the folder VelocityOutput
 so these files have a place to go.
Now run the simulation again.

Now if you open VelocityOutput you will see 11 files!
We can use those to view the particles and their movements.
--Open FlowVelocityDistributionAnim.m in Matlab and run

A file called SINATRA_gif.gif will be
 created and you can see your particles!
However, it's a mess and they don't seem to move. Let's change that.
In Complex_Input
--Change the Time Step to 1e-5
```

--Change the Total Simulation Time to 10e-5.
This will allow the particles to move further each frame.
Now let's make less particles.
--Set the Number Density to 1e-22.
Now rerun the simlulation and analysis
 and boom! Particles in a box.

Step 4 - GUI

Now I'm going to let you play with the GUI. Open up SINATRA_GUI.exe.
We're going to have to change a few things.
SINATRA Output file - SINATRA_OUTPUT.txt
Tecplot Output   - SINATRA_uniform_properties.plt
Velocity Output   - VelocityOutput/velocityFromParticles_
Mesh File   - FE_NodeLocations_UnitCubeQuad8Cells_TEC.plt
Input File   - Complex_Input.txt

This GUI prints input files.
Try it by pressing "Write File" then "Open File".
Then change something and see how it changes the input file.
Finally, you can press "Simulate" to run that input file on DSMC.exe.

Have fun!

Appendix D

INPUT FILE GUIDE

This is the Input File Guide. It can be found in the ReadMe for the resources file. It explains each command of the input file and how it works.

```
Input file Guide

SINATRA reads input files line by line.
It searches for key words and ignores words which don't fit them.
Then it takes the items in the line under the keyword and
 parses that for the simulation.

There are 6 sections to the input file. The sections are
 deliminated by *
After the first section, the order of the sections does not matter.
The order of the trigger words do not matter within the sections

Section 1 - This is the intro section. It does not have
 a header trigger

"Mesh Input Filename"                                - path name to
 the file which holds the mesh information
"Number of Real Particles to Simulation Particles"  - number as a
 double which is the ratio of real particles per simulation particles
"Boundary Conditions"                                - boundary
 conditions of the walls. 6 space deliminated string arguments
  for the type of the walls. Options are "INFLOW","OUTFLOW",
   "SWALL","DWALL","PWALL". Order of the walls is
    -X,+X,-Y,+Y,-Z,+Z.
"Collision Scheme"                                   - an integer
 which signifies which collision scheme to use
"Sphere Model"                                       - an integer
 which signifies which sphere model to use
"Total Simulation Time"                              - a double
 showing the total time the simulation should run
"Time Step"                                          - a double of
 the amount of time for each time step
```

```
Section 2 - Trigger "Initial Conditions"
This section contains simple initial conditions

"Number Density"     - a double with the number density
 for the whole simulation
"Mixture"            - a space separated line of alternating
 integers and doubles. The first number is the species
  id and the second is the percentage it is in the
   mixture being simulated
"Stream Temperature" - a double temperature in Kelvin
"Stream Velocity"    - space separated doubles which are
 the X,Y,Z direction of the stream velocity


Section 3 - Trigger "Boundary Condition Information"
This section contains the information about the Boundaries

There are 6 items which should be put in this.
 They each start with "BC X".
X is the number of the boundary from 0 to 5
 following -X,+X,-Y,+Y,-Z,+Z.
Each section ends with a "&".
They contain

"Number Density"
"Mixture"
"Stream Temperature"
"Stream Velocity"

where each are defined the same way as in Section 2.


Section 4 - Trigger "Output Information"
This section defined the output variables needed to view the data

"Output File Name"           - a string path to
 where the simple output file will be placed
"Tecplot Base Name"          - a string path which
 is the base of the Tecplot output files. It will be
  edited with timestep information for each new file.
   WARNING - if the folder path does not exist, no
    files will be created.
"Sample Cell Type"           - a flag for which sampling
 should be used. 1 stands for leaf cells and 2 stands for
```

```
  the immediate parents of the leaf cells
"Tecplot Sample Frequency"   - an integer of the number of
 time steps between each sampling
"Tecplot Sample Start Time"  - a double which defines the
 start time for Tecplot sampling
"Particle Animation"         - a three lined argument.
 first is start time. second is frequency (integer), and
  final is the base file name for the animation files



Section 5 - Trigger "Species Information"
This section holds the information for the species to be used

Each species section starts with the trigger "Species"
After this a list of items are included which are added
 to that species class
These are listed below in order.

Reference Temperature            - Kelvin    - double
Molecular Diameter               - meters    - double
Mass                             - kilograms - double
Rotational Degrees of Freedom    - number    - integer
Vibrational Degrees of Freedom   - number    - integer
Characteristic Temperature (Rot) - Kelvin    - double
Characteristic Temperature (Vib) - Kelvin    - double
Viscosity Index                  - number    - double
Viscosity Coefficient            - number    - double
Charge                           - Columbs   - double
VSS Exponent (VSS sphere only)   - number    - double


Section 6 - Trigger "Optional Keywords"
Optional items for the simulation.
These don't necessarily have a second line, they can be
 used just as single line triggers

"diffusion"              - if trigger is there, enables
 the diffusion test case
"init_velocity_gradient" - Next line is the direction of
 the gradient (X,Y,Z). Next line is 6 item deliminated by
  spaces with the inputs for the velocity gradient in the
   normal 6 direction order
"Charged Simulation"     - triggers a charged simulation
"Parallel Enabled"       - triggers a parallel simulation
```

```
Any other questions about the input file can be solved
 by emailing the original developers. Or by reading
  through input.cpp and input.h.
```

Appendix E

GAUSS-SEIDEL SOLVER VERIFICATION

Below are the outputs of the Gauss-Seidel solvers. They are printed out in order of the nodes where the nodes are 2D flattened arrays of 3D arrays. These are 64 cells and therefore a $5 \times 5 \times 5$ node structure. The first number, Rhs, stands for b_i. The second, below, stands for the dot product of the stencil below that index times the potential below that index. Above is the dot product of the stencil above that index times the potential above that index. The next one, (u,u), is the value of the stencil at that index. Finally, phi is the electric potential at that index calculated through $Rhs - below - above)/(u, u)$.

The solvers show exactly the same calculations when given the same initial conditions. While this is hard to see in this format it is trivial when the logs are analyzed with a text comparison tool.

SINATRA Solver

```
Rhs:0 below:0 above:0 (u,u):-6 phi:-0
Rhs:0 below:0 above:0 (u,u):-36 phi:-0
Rhs:0 below:0 above:0 (u,u):-36 phi:-0
Rhs:0 below:0 above:0 (u,u):-36 phi:-0
Rhs:0 below:0 above:0 (u,u):-6 phi:-0
Rhs:0 below:0 above:0 (u,u):-36 phi:-0
Rhs:0 below:0 above:0 (u,u):-66 phi:-0
Rhs:0 below:0 above:0 (u,u):-66 phi:-0
Rhs:0 below:0 above:0 (u,u):-66 phi:-0
Rhs:0 below:0 above:0 (u,u):-36 phi:-0
Rhs:0 below:0 above:0 (u,u):-36 phi:-0
Rhs:0 below:0 above:0 (u,u):-66 phi:-0
Rhs:0 below:0 above:0 (u,u):-66 phi:-0
Rhs:0 below:0 above:0 (u,u):-66 phi:-0
```

```
Rhs:0 below:0 above:0 (u,u):-36 phi:-0
Rhs:0 below:0 above:0 (u,u):-36 phi:-0
Rhs:0 below:0 above:0 (u,u):-66 phi:-0
Rhs:0 below:0 above:0 (u,u):-66 phi:-0
Rhs:0 below:0 above:0 (u,u):-66 phi:-0
Rhs:0 below:0 above:0 (u,u):-36 phi:-0
Rhs:0 below:0 above:0 (u,u):-6 phi:-0
Rhs:0 below:0 above:0 (u,u):-36 phi:-0
Rhs:0 below:0 above:0 (u,u):-36 phi:-0
Rhs:0 below:0 above:0 (u,u):-36 phi:-0
Rhs:0 below:0 above:0 (u,u):-6 phi:-0
Rhs:0 below:0 above:0 (u,u):-36 phi:-0
Rhs:0 below:0 above:0 (u,u):-66 phi:-0
Rhs:0 below:0 above:0 (u,u):-66 phi:-0
Rhs:0 below:0 above:0 (u,u):-66 phi:-0
Rhs:0 below:0 above:0 (u,u):-36 phi:-0
Rhs:0 below:0 above:0 (u,u):-66 phi:-0
Rhs:1.81e+04 below:0 above:0 (u,u):-96 phi:-188
Rhs:1.81e+04 below:-3.02e+03 above:0 (u,u):-96 phi:-220
Rhs:1.81e+04 below:-3.52e+03 above:0 (u,u):-96 phi:-225
Rhs:0 below:-450 above:0 (u,u):-66 phi:-6.82
Rhs:0 below:0 above:0 (u,u):-66 phi:-0
Rhs:1.81e+04 below:-3.02e+03 above:0 (u,u):-96 phi:-220
Rhs:1.81e+04 below:-7.04e+03 above:0 (u,u):-96 phi:-262
Rhs:1.81e+04 below:-7.79e+03 above:0 (u,u):-96 phi:-270
Rhs:0 below:-648 above:0 (u,u):-66 phi:-9.82
Rhs:0 below:0 above:0 (u,u):-66 phi:-0
Rhs:1.81e+04 below:-3.52e+03 above:0 (u,u):-96 phi:-225
Rhs:1.81e+04 below:-7.79e+03 above:0 (u,u):-96 phi:-270
Rhs:1.81e+04 below:-8.63e+03 above:0 (u,u):-96 phi:-278
Rhs:0 below:-714 above:0 (u,u):-66 phi:-10.8
Rhs:0 below:0 above:0 (u,u):-36 phi:-0
Rhs:0 below:-450 above:0 (u,u):-66 phi:-6.82
Rhs:0 below:-648 above:0 (u,u):-66 phi:-9.82
Rhs:0 below:-714 above:0 (u,u):-66 phi:-10.8
Rhs:0 below:-43.3 above:0 (u,u):-36 phi:-1.2
Rhs:0 below:0 above:0 (u,u):-36 phi:-0
Rhs:0 below:0 above:0 (u,u):-66 phi:-0
Rhs:0 below:0 above:0 (u,u):-66 phi:-0
Rhs:0 below:0 above:0 (u,u):-66 phi:-0
Rhs:0 below:0 above:0 (u,u):-36 phi:-0
Rhs:0 below:0 above:0 (u,u):-66 phi:-0
Rhs:1.81e+04 below:-3.02e+03 above:0 (u,u):-96 phi:-220
Rhs:1.81e+04 below:-7.04e+03 above:0 (u,u):-96 phi:-262
Rhs:1.81e+04 below:-7.79e+03 above:0 (u,u):-96 phi:-270
```

```
Rhs:0 below:-648 above:0 (u,u):-66 phi:-9.82
Rhs:0 below:0 above:0 (u,u):-66 phi:-0
Rhs:1.81e+04 below:-7.04e+03 above:0 (u,u):-96 phi:-262
Rhs:1.81e+04 below:-1.26e+04 above:0 (u,u):-96 phi:-319
Rhs:1.81e+04 below:-1.37e+04 above:0 (u,u):-96 phi:-332
Rhs:0 below:-978 above:0 (u,u):-66 phi:-14.8
Rhs:0 below:0 above:0 (u,u):-66 phi:-0
Rhs:1.81e+04 below:-7.79e+03 above:0 (u,u):-96 phi:-270
Rhs:1.81e+04 below:-1.37e+04 above:0 (u,u):-96 phi:-332
Rhs:1.81e+04 below:-1.51e+04 above:0 (u,u):-96 phi:-345
Rhs:0 below:-1.1e+03 above:0 (u,u):-66 phi:-16.7
Rhs:0 below:0 above:0 (u,u):-36 phi:-0
Rhs:0 below:-648 above:0 (u,u):-66 phi:-9.82
Rhs:0 below:-978 above:0 (u,u):-66 phi:-14.8
Rhs:0 below:-1.1e+03 above:0 (u,u):-66 phi:-16.7
Rhs:0 below:-85.9 above:0 (u,u):-36 phi:-2.39
Rhs:0 below:0 above:0 (u,u):-36 phi:-0
Rhs:0 below:0 above:0 (u,u):-66 phi:-0
Rhs:0 below:0 above:0 (u,u):-66 phi:-0
Rhs:0 below:0 above:0 (u,u):-66 phi:-0
Rhs:0 below:0 above:0 (u,u):-36 phi:-0
Rhs:0 below:0 above:0 (u,u):-66 phi:-0
Rhs:1.81e+04 below:-3.52e+03 above:0 (u,u):-96 phi:-225
Rhs:1.81e+04 below:-7.79e+03 above:0 (u,u):-96 phi:-270
Rhs:1.81e+04 below:-8.63e+03 above:0 (u,u):-96 phi:-278
Rhs:0 below:-714 above:0 (u,u):-66 phi:-10.8
Rhs:0 below:0 above:0 (u,u):-66 phi:-0
Rhs:1.81e+04 below:-7.79e+03 above:0 (u,u):-96 phi:-270
Rhs:1.81e+04 below:-1.37e+04 above:0 (u,u):-96 phi:-332
Rhs:1.81e+04 below:-1.51e+04 above:0 (u,u):-96 phi:-345
Rhs:0 below:-1.1e+03 above:0 (u,u):-66 phi:-16.7
Rhs:0 below:0 above:0 (u,u):-66 phi:-0
Rhs:1.81e+04 below:-8.63e+03 above:0 (u,u):-96 phi:-278
Rhs:1.81e+04 below:-1.51e+04 above:0 (u,u):-96 phi:-345
Rhs:1.81e+04 below:-1.66e+04 above:0 (u,u):-96 phi:-361
Rhs:0 below:-1.26e+03 above:0 (u,u):-66 phi:-19
Rhs:0 below:0 above:0 (u,u):-36 phi:-0
Rhs:0 below:-714 above:0 (u,u):-66 phi:-10.8
Rhs:0 below:-1.1e+03 above:0 (u,u):-66 phi:-16.7
Rhs:0 below:-1.26e+03 above:0 (u,u):-66 phi:-19
Rhs:0 below:-114 above:0 (u,u):-36 phi:-3.18
Rhs:0 below:0 above:0 (u,u):-6 phi:-0
Rhs:0 below:0 above:0 (u,u):-36 phi:-0
Rhs:0 below:0 above:0 (u,u):-36 phi:-0
Rhs:0 below:0 above:0 (u,u):-36 phi:-0
```

```
Rhs:0 below:0 above:0 (u,u):-6 phi:-0
Rhs:0 below:0 above:0 (u,u):-36 phi:-0
Rhs:0 below:-450 above:0 (u,u):-66 phi:-6.82
Rhs:0 below:-648 above:0 (u,u):-66 phi:-9.82
Rhs:0 below:-714 above:0 (u,u):-66 phi:-10.8
Rhs:0 below:-43.3 above:0 (u,u):-36 phi:-1.2
Rhs:0 below:0 above:0 (u,u):-36 phi:-0
Rhs:0 below:-648 above:0 (u,u):-66 phi:-9.82
Rhs:0 below:-978 above:0 (u,u):-66 phi:-14.8
Rhs:0 below:-1.1e+03 above:0 (u,u):-66 phi:-16.7
Rhs:0 below:-85.9 above:0 (u,u):-36 phi:-2.39
Rhs:0 below:0 above:0 (u,u):-36 phi:-0
Rhs:0 below:-714 above:0 (u,u):-66 phi:-10.8
Rhs:0 below:-1.1e+03 above:0 (u,u):-66 phi:-16.7
Rhs:0 below:-1.26e+03 above:0 (u,u):-66 phi:-19
Rhs:0 below:-114 above:0 (u,u):-36 phi:-3.18
Rhs:0 below:0 above:0 (u,u):-6 phi:-0
Rhs:0 below:-43.3 above:0 (u,u):-36 phi:-1.2
Rhs:0 below:-85.9 above:0 (u,u):-36 phi:-2.39
Rhs:0 below:-114 above:0 (u,u):-36 phi:-3.18
Rhs:0 below:-19.1 above:0 (u,u):-6 phi:-3.18
Iteration 0 with error of 5.29e+04
```

Verified Solver

```
Rhs:0 below:0 above:0 (u,u):-6 phi:-0
Rhs:0 below:-0 above:0 (u,u):-36 phi:-0
Rhs:0 below:0 above:0 (u,u):-36 phi:-0
Rhs:0 below:0 above:0 (u,u):-36 phi:-0
Rhs:0 below:0 above:0 (u,u):-6 phi:-0
Rhs:0 below:0 above:0 (u,u):-36 phi:-0
Rhs:0 below:0 above:0 (u,u):-66 phi:-0
Rhs:0 below:0 above:0 (u,u):-66 phi:-0
Rhs:0 below:0 above:0 (u,u):-66 phi:-0
Rhs:0 below:0 above:0 (u,u):-36 phi:-0
Rhs:0 below:0 above:0 (u,u):-36 phi:-0
Rhs:0 below:0 above:0 (u,u):-66 phi:-0
Rhs:0 below:0 above:0 (u,u):-66 phi:-0
Rhs:0 below:0 above:0 (u,u):-66 phi:-0
Rhs:0 below:0 above:0 (u,u):-36 phi:-0
Rhs:0 below:0 above:0 (u,u):-36 phi:-0
Rhs:0 below:0 above:0 (u,u):-66 phi:-0
```

```
Rhs:0 below:0 above:0 (u,u):-66 phi:-0
Rhs:0 below:0 above:0 (u,u):-66 phi:-0
Rhs:0 below:0 above:0 (u,u):-36 phi:-0
Rhs:0 below:0 above:0 (u,u):-6 phi:-0
Rhs:0 below:0 above:0 (u,u):-36 phi:-0
Rhs:0 below:0 above:0 (u,u):-36 phi:-0
Rhs:0 below:0 above:0 (u,u):-36 phi:-0
Rhs:0 below:0 above:0 (u,u):-6 phi:-0
Rhs:0 below:0 above:0 (u,u):-36 phi:-0
Rhs:0 below:0 above:0 (u,u):-66 phi:-0
Rhs:0 below:0 above:0 (u,u):-66 phi:-0
Rhs:0 below:0 above:0 (u,u):-66 phi:-0
Rhs:0 below:0 above:0 (u,u):-36 phi:-0
Rhs:0 below:0 above:0 (u,u):-66 phi:-0
Rhs:1.81e+04 below:0 above:0 (u,u):-96 phi:-188
Rhs:1.81e+04 below:-3.02e+03 above:0 (u,u):-96 phi:-220
Rhs:1.81e+04 below:-3.52e+03 above:0 (u,u):-96 phi:-225
Rhs:0 below:-450 above:0 (u,u):-66 phi:-6.82
Rhs:0 below:0 above:0 (u,u):-66 phi:-0
Rhs:1.81e+04 below:-3.02e+03 above:0 (u,u):-96 phi:-220
Rhs:1.81e+04 below:-7.04e+03 above:0 (u,u):-96 phi:-262
Rhs:1.81e+04 below:-7.79e+03 above:0 (u,u):-96 phi:-270
Rhs:0 below:-648 above:0 (u,u):-66 phi:-9.82
Rhs:0 below:0 above:0 (u,u):-66 phi:-0
Rhs:1.81e+04 below:-3.52e+03 above:0 (u,u):-96 phi:-225
Rhs:1.81e+04 below:-7.79e+03 above:0 (u,u):-96 phi:-270
Rhs:1.81e+04 below:-8.63e+03 above:0 (u,u):-96 phi:-278
Rhs:0 below:-714 above:0 (u,u):-66 phi:-10.8
Rhs:0 below:0 above:0 (u,u):-36 phi:-0
Rhs:0 below:-450 above:0 (u,u):-66 phi:-6.82
Rhs:0 below:-648 above:0 (u,u):-66 phi:-9.82
Rhs:0 below:-714 above:0 (u,u):-66 phi:-10.8
Rhs:0 below:-43.3 above:0 (u,u):-36 phi:-1.2
Rhs:0 below:0 above:0 (u,u):-36 phi:-0
Rhs:0 below:0 above:0 (u,u):-66 phi:-0
Rhs:0 below:0 above:0 (u,u):-66 phi:-0
Rhs:0 below:0 above:0 (u,u):-66 phi:-0
Rhs:0 below:0 above:0 (u,u):-36 phi:-0
Rhs:0 below:0 above:0 (u,u):-66 phi:-0
Rhs:1.81e+04 below:-3.02e+03 above:0 (u,u):-96 phi:-220
Rhs:1.81e+04 below:-7.04e+03 above:0 (u,u):-96 phi:-262
Rhs:1.81e+04 below:-7.79e+03 above:0 (u,u):-96 phi:-270
Rhs:0 below:-648 above:0 (u,u):-66 phi:-9.82
Rhs:0 below:0 above:0 (u,u):-66 phi:-0
Rhs:1.81e+04 below:-7.04e+03 above:0 (u,u):-96 phi:-262
```

```
Rhs:1.81e+04 below:-1.26e+04 above:0 (u,u):-96 phi:-319
Rhs:1.81e+04 below:-1.37e+04 above:0 (u,u):-96 phi:-332
Rhs:0 below:-977 above:0 (u,u):-66 phi:-14.8
Rhs:0 below:0 above:0 (u,u):-66 phi:-0
Rhs:1.81e+04 below:-7.79e+03 above:0 (u,u):-96 phi:-270
Rhs:1.81e+04 below:-1.37e+04 above:0 (u,u):-96 phi:-332
Rhs:1.81e+04 below:-1.51e+04 above:0 (u,u):-96 phi:-345
Rhs:0 below:-1.1e+03 above:0 (u,u):-66 phi:-16.7
Rhs:0 below:0 above:0 (u,u):-36 phi:-0
Rhs:0 below:-648 above:0 (u,u):-66 phi:-9.82
Rhs:0 below:-977 above:0 (u,u):-66 phi:-14.8
Rhs:0 below:-1.1e+03 above:0 (u,u):-66 phi:-16.7
Rhs:0 below:-85.9 above:0 (u,u):-36 phi:-2.39
Rhs:0 below:0 above:0 (u,u):-36 phi:-0
Rhs:0 below:0 above:0 (u,u):-66 phi:-0
Rhs:0 below:0 above:0 (u,u):-66 phi:-0
Rhs:0 below:0 above:0 (u,u):-66 phi:-0
Rhs:0 below:0 above:0 (u,u):-36 phi:-0
Rhs:0 below:0 above:0 (u,u):-66 phi:-0
Rhs:1.81e+04 below:-3.52e+03 above:0 (u,u):-96 phi:-225
Rhs:1.81e+04 below:-7.79e+03 above:0 (u,u):-96 phi:-270
Rhs:1.81e+04 below:-8.63e+03 above:0 (u,u):-96 phi:-278
Rhs:0 below:-714 above:0 (u,u):-66 phi:-10.8
Rhs:0 below:0 above:0 (u,u):-66 phi:-0
Rhs:1.81e+04 below:-7.79e+03 above:0 (u,u):-96 phi:-270
Rhs:1.81e+04 below:-1.37e+04 above:0 (u,u):-96 phi:-332
Rhs:1.81e+04 below:-1.51e+04 above:0 (u,u):-96 phi:-345
Rhs:0 below:-1.1e+03 above:0 (u,u):-66 phi:-16.7
Rhs:0 below:0 above:0 (u,u):-66 phi:-0
Rhs:1.81e+04 below:-8.63e+03 above:0 (u,u):-96 phi:-278
Rhs:1.81e+04 below:-1.51e+04 above:0 (u,u):-96 phi:-345
Rhs:1.81e+04 below:-1.66e+04 above:0 (u,u):-96 phi:-361
Rhs:0 below:-1.26e+03 above:0 (u,u):-66 phi:-19
Rhs:0 below:0 above:0 (u,u):-36 phi:-0
Rhs:0 below:-714 above:0 (u,u):-66 phi:-10.8
Rhs:0 below:-1.1e+03 above:0 (u,u):-66 phi:-16.7
Rhs:0 below:-1.26e+03 above:0 (u,u):-66 phi:-19
Rhs:0 below:-114 above:0 (u,u):-36 phi:-3.18
Rhs:0 below:0 above:0 (u,u):-6 phi:-0
Rhs:0 below:0 above:0 (u,u):-36 phi:-0
Rhs:0 below:0 above:0 (u,u):-36 phi:-0
Rhs:0 below:0 above:0 (u,u):-36 phi:-0
Rhs:0 below:0 above:0 (u,u):-6 phi:-0
Rhs:0 below:0 above:0 (u,u):-36 phi:-0
Rhs:0 below:-450 above:0 (u,u):-66 phi:-6.82
```

```
Rhs:0 below:-648 above:0 (u,u):-66 phi:-9.82
Rhs:0 below:-714 above:0 (u,u):-66 phi:-10.8
Rhs:0 below:-43.3 above:0 (u,u):-36 phi:-1.2
Rhs:0 below:0 above:0 (u,u):-36 phi:-0
Rhs:0 below:-648 above:0 (u,u):-66 phi:-9.82
Rhs:0 below:-977 above:0 (u,u):-66 phi:-14.8
Rhs:0 below:-1.1e+03 above:0 (u,u):-66 phi:-16.7
Rhs:0 below:-85.9 above:0 (u,u):-36 phi:-2.39
Rhs:0 below:0 above:0 (u,u):-36 phi:-0
Rhs:0 below:-714 above:0 (u,u):-66 phi:-10.8
Rhs:0 below:-1.1e+03 above:0 (u,u):-66 phi:-16.7
Rhs:0 below:-1.26e+03 above:0 (u,u):-66 phi:-19
Rhs:0 below:-114 above:0 (u,u):-36 phi:-3.18
Rhs:0 below:0 above:0 (u,u):-6 phi:-0
Rhs:0 below:-43.3 above:0 (u,u):-36 phi:-1.2
Rhs:0 below:-85.9 above:0 (u,u):-36 phi:-2.39
Rhs:0 below:-114 above:0 (u,u):-36 phi:-3.18
Rhs:0 below:-19.1 above:0 (u,u):-6 phi:-3.18
1,5.287557e+04
```

Appendix F

SINATRA INPUT FILES FOR VALIDATION

Cell Convergence Visual

```
*** SINATRA Input File ***
*** Created by User domin with SINATRA_GUI.m ***
*** 09-May-2019 12:23:24 ***

Mesh Input Filename
../resources/FE_NodeLocations_UnitCubeQuad64Cells_TEC.plt

Number of Real Particles to Simulation Particles
1e25

Boundary Conditions
OUTFLOW OUTFLOW OUTFLOW OUTFLOW OUTFLOW OUTFLOW

Collision Scheme
0

Sphere Model
1

Total Simulation Time
1e-5

Time Step
1e-5

Initial Conditions

Number Density
1e28

Mixture
1 1.0

Stream Temperature
100
```

```
Stream Velocity
0 0 0

***

Boundary Condition Information

BC 0

Number Density
1e+23

Mixture
1 1

Stream Temperature
300

Stream Velocity
0 0 0

&

BC 1

Number Density
1e+23

Mixture
1 1

Stream Temperature
300

Stream Velocity
0 0 0

&

BC 2

Number Density
1e+23
```

```
Mixture
1 1

Stream Temperature
300

Stream Velocity
0 0 0

&

BC 3

Number Density
1e+23

Mixture
1 1

Stream Temperature
300

Stream Velocity
0 0 0

&

BC 4

Number Density
1e+23

Mixture
1 1

Stream Temperature
300

Stream Velocity
0 0 0

&

BC 5
```

```
Number Density
1e+23

Mixture
1 1

Stream Temperature
300

Stream Velocity
0 0 0

***

Output Information

Output File Name
../resources/SINATRA_OUTPUT.txt

Tecplot Base Name
../resources/TecplotOutput/SINATRA_uniform_properties.plt

Sample Cell Type
1

Tecplot Sample Frequency
1

Particle Animation
0
1
../resources/VelocityOutput/velocityFromParticles_

***

Species Information

Species 1 (Nitrogen, N2)
273
4.17e-010
4.65e-026
2
2
2.88
3371
```

```
0.74
1.656e-05
1e-19

Species 2 (Oxygen, O2)
273
4.07e-010
5.31e-026
2
2
2.07
2256
0.77
1.919e-05
0

Species 3 (Argon, Ar)
273
4.17e-010
6.63e-026
0
0
0
0
0.81
2.117e-05
0

Species 4 (Carbon Dioxide, CO2)
273
5.62e-010
7.31e-026
3
4
0.561
1700
0.93
1.380e-05
0

***

Optional Keywords

Charged Simulation
```

***

Ambipolar - 5% solver

```
*** SINATRA Input File ***

Mesh Input Filename
../resources/FE_NodeLocations_UnitCubeQuad4096Cells_TEC.plt

Number of Real Particles to Simulation Particles
1e9

Boundary Conditions
OUTFLOW OUTFLOW OUTFLOW OUTFLOW OUTFLOW OUTFLOW

Collision Scheme
0

Sphere Model
1

Total Simulation Time
500e-7

Time Step
1e-7

Initial Conditions

Number Density
1e12

Mixture
3 1.0

Stream Temperature
5000

Stream Velocity
0 0 0

***

Boundary Condition Information

BC 0
```

```
Number Density
1e12

Mixture
3 1.0

Stream Temperature
5000

Stream Velocity
0 0 0

&

BC 1

Number Density
1e12

Mixture
3 1.0

Stream Temperature
5000

Stream Velocity
0 0 0

&

BC 2

Number Density
1e12

Mixture
3 1.0

Stream Temperature
5000

Stream Velocity
0 0 0
```

```
&

BC 3

Number Density
1e12

Mixture
3 1.0

Stream Temperature
5000

Stream Velocity
0 0 0

&

BC 4

Number Density
1e12

Mixture
3 1.0

Stream Temperature
5000

Stream Velocity
0 0 0

&

BC 5

Number Density
1e12

Mixture
3 1.0

Stream Temperature
5000
```

```
Stream Velocity
0 0 0

***

Output Information

Output File Name
../resources/SINATRA_OUTPUT_Ambipolar_0.05.txt

Tecplot Base Name
../resources/TecplotOutput/SINATRA_Ambipolar_0.05_uniform_properties.plt

Sample Cell Type
1

Tecplot Sample Frequency
5

***

Species Information

Species 1 (Nitrogen, N2)
273
4.17e-010
4.65e-026
2
2
2.88
3371
0.74
1.656e-05
0

Species 2 (Oxygen, O2)
273
4.07e-010
5.31e-026
2
2
2.07
2256
0.77
1.919e-05
```

```
0

Species 3 (Argon, Ar)
273
4.17e-010
6.63e-026
0
0
0
0
0.81
2.117e-05
1.6e-19

Species 4 (Carbon Dioxide, CO2)
273
5.62e-010
7.31e-026
3
4
0.561
1700
0.93
1.380e-05
0

***

Optional Keywords

Parallel Enabled

Charged Simulation

***
```

Ambipolar - 64 cells

```
*** SINATRA Input File ***

Mesh Input Filename
../resources/FE_NodeLocations_UnitCubeQuad64Cells_TEC.plt

Number of Real Particles to Simulation Particles
1e9

Boundary Conditions
OUTFLOW OUTFLOW OUTFLOW OUTFLOW OUTFLOW OUTFLOW

Collision Scheme
0

Sphere Model
1

Total Simulation Time
500e-7

Time Step
1e-7

Initial Conditions

Number Density
1e12

Mixture
3 1.0

Stream Temperature
5000

Stream Velocity
0 0 0

***

Boundary Condition Information

BC 0
```

```
Number Density
1e12

Mixture
3 1.0

Stream Temperature
5000

Stream Velocity
0 0 0

&

BC 1

Number Density
1e12

Mixture
3 1.0

Stream Temperature
5000

Stream Velocity
0 0 0

&

BC 2

Number Density
1e12

Mixture
3 1.0

Stream Temperature
5000

Stream Velocity
0 0 0
```

```
&

BC 3

Number Density
1e12

Mixture
3 1.0

Stream Temperature
5000

Stream Velocity
0 0 0

&

BC 4

Number Density
1e12

Mixture
3 1.0

Stream Temperature
5000

Stream Velocity
0 0 0

&

BC 5

Number Density
1e12

Mixture
3 1.0

Stream Temperature
5000
```

```
Stream Velocity
0 0 0

***

Output Information

Output File Name
../resources/SINATRA_OUTPUT_Ambipolar_64.txt

Tecplot Base Name
../resources/TecplotOutput/SINATRA_Ambipolar_64_uniform_properties.plt

Sample Cell Type
1

Tecplot Sample Frequency
5

***

Species Information

Species 1 (Nitrogen, N2)
273
4.17e-010
4.65e-026
2
2
2.88
3371
0.74
1.656e-05
0

Species 2 (Oxygen, O2)
273
4.07e-010
5.31e-026
2
2
2.07
2256
0.77
1.919e-05
```

```
0

Species 3 (Argon, Ar)
273
4.17e-010
6.63e-026
0
0
0
0
0.81
2.117e-05
1.6e-19

Species 4 (Carbon Dioxide, CO2)
273
5.62e-010
7.31e-026
3
4
0.561
1700
0.93
1.380e-05
0

***

Optional Keywords

Parallel Enabled

Charged Simulation

***
```

Ambipolar - 10 times more particles

```
*** SINATRA Input File ***

Mesh Input Filename
../resources/FE_NodeLocations_UnitCubeQuad64Cells_TEC.plt

Number of Real Particles to Simulation Particles
1e6

Boundary Conditions
OUTFLOW OUTFLOW OUTFLOW OUTFLOW OUTFLOW OUTFLOW

Collision Scheme
0

Sphere Model
1

Total Simulation Time
500e-7

Time Step
1e-7

Initial Conditions

Number Density
1e12

Mixture
3 1.0

Stream Temperature
5000

Stream Velocity
0 0 0

***

Boundary Condition Information

BC 0
```

```
Number Density
1e12

Mixture
3 1.0

Stream Temperature
5000

Stream Velocity
0 0 0

&

BC 1

Number Density
1e12

Mixture
3 1.0

Stream Temperature
5000

Stream Velocity
0 0 0

&

BC 2

Number Density
1e12

Mixture
3 1.0

Stream Temperature
5000

Stream Velocity
0 0 0
```

```
&

BC 3

Number Density
1e12

Mixture
3 1.0

Stream Temperature
5000

Stream Velocity
0 0 0

&

BC 4

Number Density
1e12

Mixture
3 1.0

Stream Temperature
5000

Stream Velocity
0 0 0

&

BC 5

Number Density
1e12

Mixture
3 1.0

Stream Temperature
5000
```

```
Stream Velocity
0 0 0

***

Output Information

Output File Name
../resources/SINATRA_OUTPUT_Ambipolar_64more.txt

Tecplot Base Name
../resources/TecplotOutput/SINATRA_Ambipolar_64more_uniform_properties.plt

Sample Cell Type
1

Tecplot Sample Frequency
5

***

Species Information

Species 1 (Nitrogen, N2)
273
4.17e-010
4.65e-026
2
2
2.88
3371
0.74
1.656e-05
0

Species 2 (Oxygen, O2)
273
4.07e-010
5.31e-026
2
2
2.07
2256
0.77
1.919e-05
```

```
0

Species 3 (Argon, Ar)
273
4.17e-010
6.63e-026
0
0
0
0
0.81
2.117e-05
1.6e-19

Species 4 (Carbon Dioxide, CO2)
273
5.62e-010
7.31e-026
3
4
0.561
1700
0.93
1.380e-05
0

***

Optional Keywords

Parallel Enabled

Charged Simulation

***
```

Neutral diffusion

```
*** SINATRA Input File ***

Mesh Input Filename
../resources/FE_NodeLocations_UnitCubeQuad4096Cells_TEC.plt

Number of Real Particles to Simulation Particles
1e9

Boundary Conditions
OUTFLOW OUTFLOW OUTFLOW OUTFLOW OUTFLOW OUTFLOW

Collision Scheme
0

Sphere Model
1

Total Simulation Time
500e-7

Time Step
1e-7

Initial Conditions

Number Density
1e12

Mixture
3 1.0

Stream Temperature
5000

Stream Velocity
0 0 0

***

Boundary Condition Information

BC 0
```

```
Number Density
1e12

Mixture
3 1.0

Stream Temperature
5000

Stream Velocity
0 0 0

&

BC 1

Number Density
1e12

Mixture
3 1.0

Stream Temperature
5000

Stream Velocity
0 0 0

&

BC 2

Number Density
1e12

Mixture
3 1.0

Stream Temperature
5000

Stream Velocity
0 0 0
```

```
&

BC 3

Number Density
1e12

Mixture
3 1.0

Stream Temperature
5000

Stream Velocity
0 0 0

&

BC 4

Number Density
1e12

Mixture
3 1.0

Stream Temperature
5000

Stream Velocity
0 0 0

&

BC 5

Number Density
1e12

Mixture
3 1.0

Stream Temperature
5000
```

```
Stream Velocity
0 0 0

***

Output Information

Output File Name
../resources/SINATRA_OUTPUT_Neutral.txt

Tecplot Base Name
../resources/TecplotOutput/SINATRA_Neutral_uniform_properties.plt

Sample Cell Type
1

Tecplot Sample Frequency
5

***

Species Information

Species 1 (Nitrogen, N2)
273
4.17e-010
4.65e-026
2
2
2.88
3371
0.74
1.656e-05
0

Species 2 (Oxygen, O2)
273
4.07e-010
5.31e-026
2
2
2.07
2256
0.77
1.919e-05
```

```
0

Species 3 (Argon, Ar)
273
4.17e-010
6.63e-026
0
0
0
0
0.81
2.117e-05
1.6e-19

Species 4 (Carbon Dioxide, CO2)
273
5.62e-010
7.31e-026
3
4
0.561
1700
0.93
1.380e-05
0

***

Optional Keywords

Parallel Enabled

***
```

Steady Flow

```
*** SINATRA Input File ***

Mesh Input Filename
../resources/FE_NodeLocations_UnitCubeQuad512Cells_TEC.plt

Number of Real Particles to Simulation Particles
0.5e9

Boundary Conditions
INFLOW OUTFLOW OUTFLOW OUTFLOW OUTFLOW OUTFLOW

Collision Scheme
0

Sphere Model
1

Total Simulation Time
100e-8

Time Step
1e-8

Initial Conditions

Number Density
1e12

Mixture
3 1.0

Stream Temperature
500

Stream Velocity
1000 0 0

***

Boundary Condition Information

BC 0
```

```
Number Density
1e12

Mixture
3 1.0

Stream Temperature
500

Stream Velocity
1000 0 0

&

BC 1

Number Density
1e12

Mixture
3 1.0

Stream Temperature
500

Stream Velocity
0 0 0

&

BC 2

Number Density
1e12

Mixture
3 1.0

Stream Temperature
500

Stream Velocity
0 0 0
```

```
&

BC 3

Number Density
1e12

Mixture
3 1.0

Stream Temperature
500

Stream Velocity
0 0 0

&

BC 4

Number Density
1e12

Mixture
3 1.0

Stream Temperature
500

Stream Velocity
0 0 0

&

BC 5

Number Density
1e12

Mixture
3 1.0

Stream Temperature
500
```

```
Stream Velocity
0 0 0

***

Output Information

Output File Name
../resources/SINATRA_OUTPUT.txt

Tecplot Base Name
../resources/TecplotOutput/SINATRA_uniform_properties.plt

Sample Cell Type
1

Tecplot Sample Frequency
500

***

Species Information

Species 1 (Nitrogen, N2)
273
4.17e-010
4.65e-026
2
2
2.88
3371
0.74
1.656e-05
0

Species 2 (Oxygen, O2)
273
4.07e-010
5.31e-026
2
2
2.07
2256
0.77
1.919e-05
```

```
0

Species 3 (Argon, Ar)
273
4.17e-010
6.63e-026
0
0
0
0
0.81
2.117e-05
1.6e-19

Species 4 (Carbon Dioxide, CO2)
273
5.62e-010
7.31e-026
3
4
0.561
1700
0.93
1.380e-05
0

***

Optional Keywords

Charged Simulation

***
```

Appendix G

AMBIPOLAR DIFFUSION COEFFICIENT

Seen below is a MATLAB^{TM} which can calculate the ambipolar diffusion coefficient.

```
% Dominic Lunde
% Validation of ambipolar diffusion

close all;
clearvars;

% page 78

%-D_a grad^2 n = 0;

e = 1.6e-19;
m_e = 9.109e-31;
m_i = 6.63e-26;
k = 1.38e-23;

n_e = 10^12;
T_eV = 100;
T_i = 5000;

% page 70
lnV = 23 - 0.5 * log(10^6 * n_e / T_eV^3);

nu = 2.9e-12 * n_e * lnV / T_eV^1.5;


mu_e = e / (m_e * nu);
mu_i = e / (m_i * nu);

D_e = T_eV / (m_e * nu);
D_i = k * T_i / (m_i * nu);

D_a = (mu_i * D_e + mu_e * D_i) / (mu_i + mu_e);
```