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

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 Mechanical Engineering

by

David Galvez

August 2018

© 2018

David Galvez

ALL RIGHTS RESERVED

COMMITTEE MEMBERSHIP

TITLE:	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
AUTHOR:	David Galvez
DATE SUBMITTED:	August 2018
COMMITTEE CHAIR:	Kim Shollenberger, Ph.D. Professor of Mechanical Engineering
COMMITTEE MEMBER:	David Marshall, Ph.D. Professor of Aerospace Engineering
COMMITTEE MEMBER:	Russell Westphal, Ph.D. Professor of Mechanical Engineering

ABSTRACT

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

David Galvez

The field of Computational Fluid Dynamics (CFD) primarily involves the approximation of the Navier-Stokes equations. However, these equations are only valid when the flow is considered continuous such that molecular interactions are abundant and predictable. The Knudsen number, Kn, which is defined as the ratio of the flow's mean free path, λ, to some characteristic length, L, quantifies the continuity of any flow, and when this parameter is large enough, alternative methods must be employed to simulate gases. The Direct Simulation Monte Carlo (DSMC) method is one which simulates rarefied gas flows by directly simulating the particles that compose the flow and using probabilistic methods to determine their collisions and properties.

This thesis discusses the development of a new DSMC simulation code, named SINATRA, which was written in object-oriented C++ and validated on Cartesian grids. The code demonstrates the ability to perform standard simulation code tasks which include reading-in a user-made input file, performing the specified simulation, and generating visualization files compatible with Tecplot 360™, a commercial post-processing software. SINATRA strategically uses an octree data structure as a storage scheme for computational grid data and uses this a backbone for particle interactions. The discussed validation cases include comparisons of initial particle properties to theoretical data, convergence studies for the sampling of macroscopic properties, and validation of transport properties through natural diffusion and Couette flow simulations. The results show successful implementation of simple DSMC procedures, and a path for future development of the code is thoroughly discussed.

ACKNOWLEDGMENTS

There are many wonderful people in this world that I would like to acknowledge, and certainly one cannot do all these people justice. First and foremost, I would like to thank my family for their love and support throughout my college experience.

I would also like to genuinely thank my committee members for their support throughout this year. Dr. Shollenberger, I am very glad you were my main advisor this year. Thank you for being there for me and providing guidance and support when I needed it. Dr. Marshall, thank you for all your time you spent with me this year and thank you for giving me the opportunity to work closely with you. I know I sort of weaseled my way into this project, so thank you for taking the risk and advising me as though I was your direct student. Dr. Westphal, your courses in Compressible Flow and Viscous Flow have not only taught me a significant amount about fluid mechanics, but also made me excited to learn more. Thank you for making school fun. Without those courses, I would be lost when learning the DSMC material.

Thank you for all my roommates and close friends throughout these past years. I am definitely blessed to have been surrounded by a large variety of people, all of whom profoundly touched my life in some way. Without you guys, I'd probably be miserable.

I am also grateful for my immediate successors on this project, Robert (Mac) Alliston and Dominic Lunde. Thank you for the weekly meetings and being there to bounce ideas off of. I wish you the best of luck in continuing this project and I can't wait to see what this code turns into down the road. I hope it treats you well.

TABLE OF CONTENTS

LIST OF TABLES ix

LIST OF FIGURES x

1 Introduction 1

1.1 Motivation 1

1.2 Literature Review 8

1.3 Thesis Goals 12

2 DSMC: Theory and Procedures 14

2.1 Overview 14

2.2 Kinetic Theory 14

2.2.1 Velocity Distribution Functions 15

2.2.2 Binary Collisions 16

2.2.3 Molecular Models 17

2.3 Basic Quantum Theory 18

2.4 Macroscopic Properties 22

2.5 Transport Properties 26

2.6 DSMC Basic Procedure 28

3 SINATRA Code Structure 29

3.1 Object-Oriented Programming and C++ 29

3.2 Cell and Particle Data Structures 31

3.2.1 Cell Structure 31

3.2.1.1 Accessing Children from Parents 39

3.2.1.2 Traversing the Octree Structure 40

3.2.1.3 Computational Domain 41

3.2.2 Particle Structure 43

3.2.3 Particle-Cell Linking 44

3.3 SINATRA Classes 45

4 SINATRA Code Implementation 47
4.1 SINATRA Simulation Overview 47
4.2 Flow Initialization 48
4.2.1 Initial Position 48
4.2.2 Initial Velocity 49
4.2.3 Initial Rotational Energy 50
4.2.4 Initial Vibrational Energy 51
4.3 Particle Advection 51
4.4 Boundary Conditions 52
4.4.1 Inlet 52
4.4.2 Outlet 53
4.4.3 Specular Wall 54
4.4.4 Diffuse Wall 54
4.4.5 Periodic Wall 55
4.5 Particle-Cell Re-Linking 55
4.6 Particle Collisions 55
4.7 Flow Property Extraction 59
4.7.1 Density 59
4.7.2 Velocity 60
4.7.3 Temperature 60
4.7.4 Pressure 62
4.8 Post Processing 63
4.8.1 Tecplot 63
4.8.2 MATLAB 63
5 Code Validation and Test Cases 64
5.1 Random Number Generator 64
5.2 Initial Flow Field Properties 65
5.2.1 Initial Position 65
5.2.2 Initial Velocity 66
5.2.3 Initial Energy 70
5.3 Property Sampling 74
5.4 Collisionless Test Cases 78

5.4.1 Statistical Scatter 78
5.4.2 Collisionless Heat Transfer 80
5.5 Molecular Diffusion 82
5.5.1 Self-Diffusion Coefficient of Argon 82
5.6 Couette Flow 86
5.6.1 Viscosity Coefficient of Argon 87
5.6.2 Prandtl Number of Nitrogen 92
6 Conclusions 95
7 Future Work 96
7.1 Initialization 97
7.2 Mesh 98
7.3 Collisions 99
7.4 Energy Considerations 99
7.5 C++: STL versus Pointers 100
7.6 Parallelization 101
7.7 Restart Files 101
7.8 Local Time Stepping 103
7.9 Addition of `Boundary` Class 103
7.10 Coupling between Sampled Properties and Stored Data 104
7.11 Simulation Stop Triggers 104
7.12 Automatic Unit Level Tests 105
7.13 DSMC Learning Curve 105
BIBLIOGRAPHY 107
APPENDIX A Basic Mesh Generator 113
APPENDIX B SINATRA Flow Chart and Classes 114
APPENDIX C SINATRA Code Snippets 115
C.1 Cell Structures 115
C.2 Property Sampling 116
APPENDIX D MATLAB Particle Animation Script 117
APPENDIX E Simulation Input Files 119
E.1 Mesh Convergence 119
E.2 Collisionless Heat Transfer 121
E.3 Self-Diffusion Flow 123
E.4 Couette Flow 125
APPENDIX F Bishop Batch Script 126

LIST OF TABLES

3.1 Relative positions of children cells. 33

3.2 Description of child cell variables in `ChildCell` struct. 35

3.3 Relative location of each index in `neighbors` array. 35

3.4 Index key for each coordinate direction. 36

3.5 Description of child cell information variables in `ChildInfo` struct. . . 36

3.6 Description of parent cell variables in `ParentCell` struct. 37

3.7 Description of particle structure variables. 43

3.8 Description of species structure variables. 44

3.9 Summary of classes in SINATRA. 46

5.1 Summary of flow properties for velocity distribution analysis. 67

5.2 Effect of the number of particles on the percent error of the average molecular speed of the system. 70

5.3 Effect of the number of particles on the average rotational energy for Nitrogen (diatomic molecule). 72

5.4 Summary of relevant species data for CO_2. 73

5.5 Effect of the number of particles on the average rotational energy for Carbon Dioxide (polyatomic molecule). 74

5.6 Prescribed properties used for mesh convergence study. 75

5.7 Comparison of percent variation between DSMC.F90 and SINATRA. 79

5.8 Diffusion coefficient results in comparison to Bird's simulations and theoretical results. 85

5.9 Viscosity Coefficient result for low speed Couette flow of Ar. 90

5.10 Transport property results for high speed Couette flow of Ar. 92

5.11 Transport property results for high speed Couette flow of N_2. 93

LIST OF FIGURES

1.1 Overview of disciplines within mechanics [1] 1
1.2 Summary of length and time scales [2]. 3
1.3 Flow regimes based on Knudsen number [3]. 4
1.4 Illustrations of (a) serial computing, and (b) parallel computing [4]. . 7
1.5 High performance computing outline [5]. 8

2.1 Depiction of a binary collision between two particles [6]. 16
2.2 Depiction of Translational Energy. 19
2.3 Depiction of Rotational Energy [7]. 20
2.4 Depiction of Vibrational Energy. 20
2.5 Schematic of energy levels for different energy modes [8]. 21
2.6 Flow Chart of General DSMC Procedures 28

3.1 Diagram of pointers [9]. 30
3.2 Tree diagram of octree data structure [10]. 32
3.3 Cell cut in Octree fashion. 33
3.4 Varying refinement from cell to cell. 34
3.5 Illustration of STL vector, called `childCells`, containing n `ChildCell` structs. 38
3.6 Illustration of STL vector, called `childInfo`, containing n `ChildInfo` structs. 38
3.7 Illustration of STL vector, called `parentCells`, containing m `ParentCell` structs. 39
3.8 Diagram of Parent-Child relationship. 40
3.9 Traversing octree data structure from root to leaf cell. 41
3.10 Most commonly used grids for SINATRA. 42
3.11 Diagram of Particle-Cell Linking 45

5.1 Histogram of a sample of 1 million pseudo-random points. 64

5.2 Projection of the initial positions of particles when viewed through a 2D plane. 66

5.3 SINATRA data of all three initial velocity components overlayed on probability distribution function. 68

5.4 SINATRA initial speed data overlayed on probability distribution function yielding a Maxwellian distribution. 69

5.5 SINATRA initial rotational energy overlayed on probability distribution function for N_2. 71

5.6 SINATRA initial rotational energy overlayed on probability distribution function for CO_2. 73

5.7 Convergence study for temperature. 75

5.8 Convergence study for pressure. 76

5.9 Convergence study for velocity. 76

5.10 Convergence study for speed. 77

5.11 The number of particles in the domain over time. 79

5.12 Simulation setup for collisionless heat transfer test case. 80

5.13 Plot of Collisionless Heat Transfer. 82

5.14 Boundary conditions used to simulate the self-diffusion of Ar. 83

5.15 Variation of number densities for both species along the x-axis of the self-diffusion of Ar. Bird's data was extracted from Figure 12.9 in [11]. 84

5.16 Variation of diffusion velocity along the x-axis of the self-diffusion of Ar. Bird's data was extracted from Figure 12.10 in [11]. 85

5.17 Boundary conditions used to simulate Couette flow (z-axis not shown to limit cluttering). 86

5.18 Velocity variation along the x-axis for low speed Couette flow of Ar. Bird's data was extracted from Figure 12.1 in [11]. 88

5.19 Temperature variation along the x-axis for low speed Couette flow of Ar. Bird's data was extracted from Figure 12.2 in [11]. 89

5.20 Density variation along the x-axis for low speed Couette flow of Ar. Bird's data was extracted from Figure 12.3 in [11]. 90

5.21 Velocity gradient along the x-axis for high speed Couette flow of Ar. Bird's data was extracted from Figure 12.7 in [11] 91

5.22 Velocity gradient along the x-axis for high speed Couette flow of N_2. . 93

B.1 SINATRA simulation flow chart with classes that perform each action. 114

CHAPTER 1

Introduction

1.1 Motivation

The term *mechanics*, in the context of engineering, is the general study of physical systems. Figure 1.1 shows an overview of the disciplines within mechanics for which engineers work.

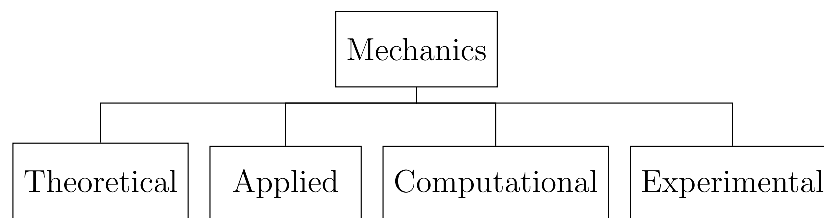


Figure 1.1 – Overview of disciplines within mechanics [1]

Computational mechanics is a branch of the general term *mechanics* concerned with the application of numerical methods and computer science techniques towards the approximate solution of physical problems. While the theory behind numerical analysis has been around since the early days of the Babylonians who numerically approximated the square root of 2 around 1600 BC, the likes of Newton and Euler significantly enhanced humanity's understanding of the art. More recently, the advent and growth of computers in the mid to late 1900s has made the widespread application of these methods possible. The rapid development of the capabilities of computers coupled with their increasingly affordable costs have made computational mechanics not only possible, but preferable in many industries ranging from agriculture, automotive, and aerospace.

Narrowing the discussion further, computational fluid dynamics (CFD), a popular branch of computational mechanics especially in the aerospace and defense industries, describes fluid flow behavior by numerically approximating the solution to the Navier-Stokes (N-S) equations. The Navier-Stokes equations, as shown below in Equation 1.1 are nonlinear partial differential equations that provide an exact description to any fluid flow field that falls into a continuum regime.

$$\frac{\partial(\rho u_i)}{\partial t} + \frac{\partial[\rho u_i u_j]}{\partial x_j} = -\frac{\partial p}{\partial x_i} + \frac{\partial \tau_{ij}}{\partial x_j} + \rho f_i \tag{1.1}$$

where p is pressure, ρ is density, x_i is a Cartesian coordinate direction, u_i is a velocity component, f_i is the sum of external forces per unit mass, and τ_{ij} is the Reynolds stress tensor. Note that Equation 1.1 follows the conventional Einstein notation where repeated indices (here i or j) are summed over the three dimensions. At the time of this writing, these equations are impossible to solve analytically except for a small subset of classical problems, and can only be solved approximately by computational tools. The details of approximating the solutions to the N-S equations are not discussed in this report, but the reader is referred to [12] for detailed derivations and discussions on the matter.

A great number of practical fluid simulation applications involve fluids that can be considered continuous and therefore, can be simulated by approximating the solution to the Navier-Stokes equations. However, there are a subset of scenarios where the continuum assumption is no longer valid and other methods must be employed.

Thus, in order for an analyst to successfully use computational tools to solve real world problems, he or she must have a fundamental understanding of the scale of the problem. Figure 1.2 shows different scales that mechanics simulations can reside in.

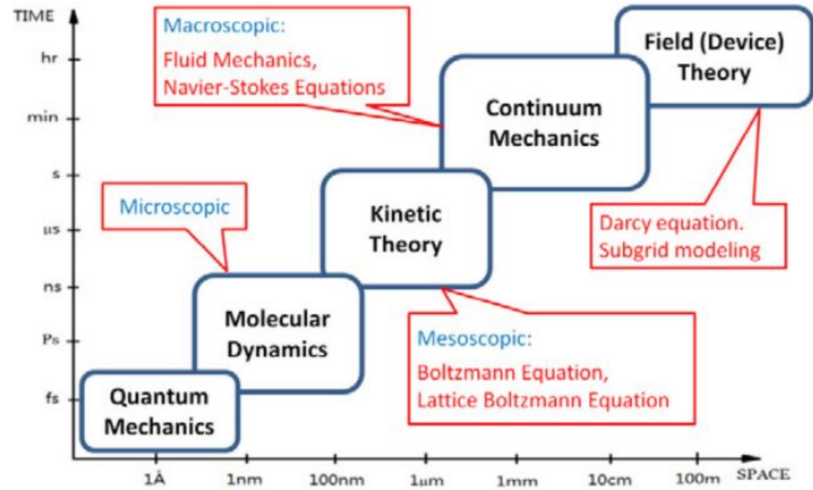


Figure 1.2 – Summary of length and time scales [2].

The parameter used to quantify the continuity of a flow is the Knudsen number, Kn, which is defined as the ratio of the molecular mean free path, λ, to a characteristic linear dimension, L.

$$Kn = \lambda / L \tag{1.2}$$

As a rule of thumb, a fluid is considered to be continuous when the Knudsen number is less than 0.1. This is the regime where the molecules that compose the fluid collide in a predictable manner. A fluid flow with a Knudsen number larger than 0.1 is considered a rarefied gas and a molecular description must be used to quantify flow properties and behavior. Note that the Knudsen number can become relatively large in one of two ways:

1. The mean free path is very large, meaning that the particles are spaced relatively far apart and collide only at a statistical frequency.
2. The characteristic length is small, meaning that the flow is traveling through or around a very small object.

The latter case occurs most often in the study of microelectromechanical systems (MEMS) where fluid travels through microchannels, while the former case describes rarefied gases that are most commonly found in the upper atmosphere. An illustration of how the flow regimes change based on the Knudsen number is shown below in Figure 1.3.

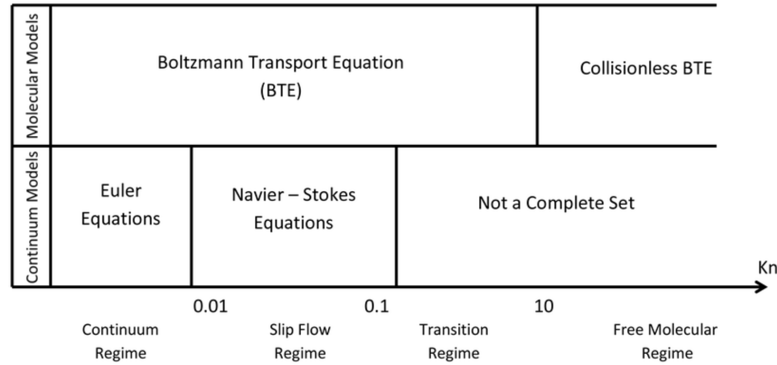


Figure 1.3 – Flow regimes based on Knudsen number [3].

This thesis will primarily focus on high Knudsen number applications relating to rarefied gases, though discussions and examples can easily be extended to both regimes.

There are several analytical ways to predict the behavior of gases in the mesoscopic and microscopic scales. One way is by modifying the boundary conditions for the solutions of the Navier-Stokes equations to account for the fact that the traditional "no-slip condition" between the gas and a surface no longer exists at higher Knudsen numbers. Although this modification has proved to work well in some model problems, it is not an all-encompassing solution. The more trusted method is by using the Boltzmann Equation (also known as the "Boltzmann Kinetic Equation" or "Boltzmann Transport Equation (BTE)"), which statistically describes the transport of atoms and molecules in a gas [13]. The Boltzmann equation is given by Equation 1.3 [14] below:

$$
\begin{aligned}
\frac{\partial f_i}{\partial t} + \mathbf{v}_i \cdot \frac{\partial f_i}{\partial \mathbf{r}} + \mathbf{F}_i \cdot \frac{\partial f_i}{\partial \mathbf{v}_i} \\
&= \sum_j \int_0^\infty \int_0^{2\pi} \int_0^\infty \Big(f_i(v_i') f_j(v_j') - f_i(v_i) f_j(v_j) \Big) g_{ij} b \mathrm{d}b \mathrm{d}\epsilon \mathrm{d}v_j \\
&= \sum_j j(f_i f_j)
\end{aligned} \tag{1.3}
$$

where $f_i(v_i)$, $f_j(v_j)$, $f_i(v_i')$, and $f_j(v_j')$ are the particle distribution functions of the ith and jth particle species before and after their collision, respectively; v_i, v_j, v_i', and v_j' are the molecular velocities of particles of the ith and jth species before and after collisions, respectively; $\mathbf{F}_i$ is the net body force on a unit mass of a particle mainly due to an electric field; g_{ij} is the initial relative velocity of the particles of the ith and jth species; ϵ is the azimuthal orientation angle of the plane where the scattering of particles takes place; and b is an impact parameter. An in depth discussion of the Boltzmann Equation and its solutions is out of the scope of this work, however related discussions of the distribution functions and other kinetic theory concepts such as the extraction of transport properties from the Boltzmann Equation are visited in Chapter 2. Similar to the N-S equations, closed form solutions only exist for a very small subset of problems after many simplifications to the equation are made [15]. However, the Boltzmann equation is not used in practical problems.

Several methods have been developed which attempt to computationally simulate fluids at a molecular level. For the sake of clarity, these methods do not attempt to simulate reality by numerically approximating well-known, well-trusted mathematical equations, but rather attempt to directly simulate the motion and interactions of molecules themselves that form the behavior which the standard equations aim to capture. The two most popular methods are the Molecular Dynamics (MD) method and the Direct Simulation Monte Carlo method (DSMC). The MD method attempts to model every individual particle in a fluid, and determine collisions and motions

deterministically. It has been shown that the MD method works best for dense gases due to the otherwise unrealistically high computational time required [16]. On the contrary, the DSMC method probabilistically simulates gases through the employment of discrete representative molecules that have properties and interact in a probabalistic nature. This is the preferred method for dilute gases since this permits the molecular motion and collisions to be uncoupled over a small time interval which allows for a more computationally efficient code. Due to its usefulness in rarefied gas dynamics analysis and, in turn, aerospace and mechanical engineering as a whole, the development of a DSMC code is worthwhile.

In addition to the motivation behind the DSMC method in general, there is a separate motivation behind this particular thesis. As noted above, the DSMC method simulates a large amount of particles, which consumes a large amount of computation time. In the summer of 2017 in a joint proposal by professors from the Aerospace, Mechanical, Electrical Engineering and Computer Science Departments, Cal Poly was granted a high performance computer (HPC), now known popularly as Bishop [17]. A high performance computer is quite simply a very powerful computer composed of many physical compute nodes that allows for efficient parallel computing, which is the ability to split up tasks into multiple processors for quicker computation time. The improvement gained from parallel computing is illustrated below in Figures 1.4a and 1.4b.

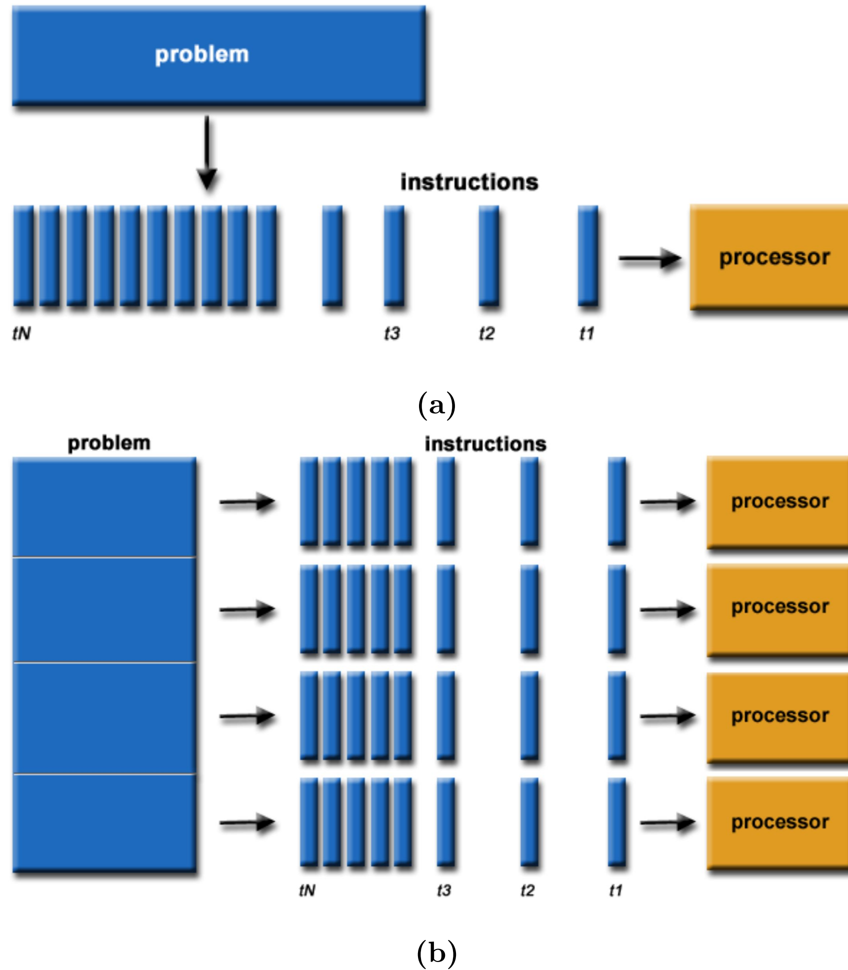


Figure 1.4 – Illustrations of (a) serial computing, and (b) parallel computing [4].

Notice how a problem run normally on a personal computer (Figure 1.4a) is now split amongst multiple processors (Figure 1.4b) and effectively completed quicker. Bishop has 240 processor cores, 1.1 Terabytes of RAM, and 12 Terabytes of scratch space for storage [18], giving it the potential to dramatically improve simulation speeds. An illustration of how high performance clusters work is shown below in Figure 1.5. With Bishop, students can access the Head Node from any terminal on campus, and when submitting simulation jobs they get sent to the compute nodes where they can be distributed among processors.

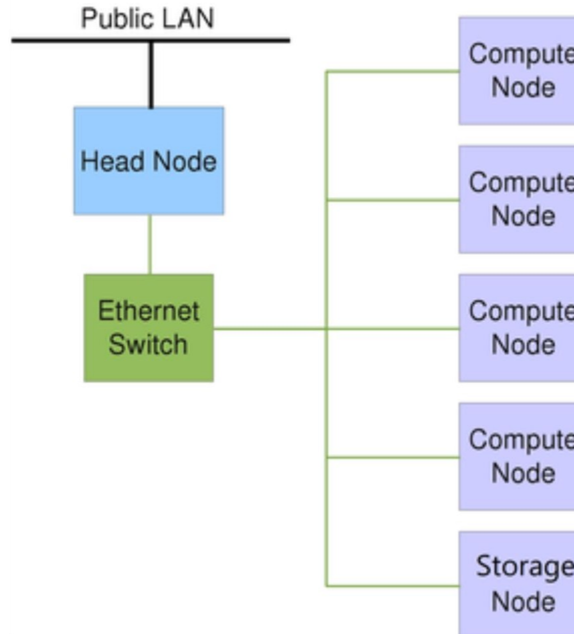


Figure 1.5 – High performance computing outline [5].

The high performance computer has already demonstrated significant improvements in speed when running commercial CFD codes. This trend is expected to be seen in other gas dynamics and physics codes (including DSMC codes) as well. Bishop has opened up new possibilities for computational simulations at Cal Poly, and has made programs initially deemed too slow possible. Now that the HPC is available to Cal Poly students, it is worth the time and effort to explore the capabilities of this new resource.

1.2 Literature Review

The DSMC method was first introduced by Professor Graeme Austin Bird in 1963 [19] at a time when computers were both slow and expensive. At that time, he primarily discussed interaction of simple molecules. However, in 1978 Bird published another paper which first utilized the Monte Carlo, or probabilistic, nature of gas flows and demonstrated successful use of the method for low-pressure rarefied gas

flow applications [3]. In 1994, Bird authored a book [11] that summarized the theory and outlined the implementation of the entire DSMC method. The book is the most cited reference for all DSMC articles and its creation marked the beginning of the DSMC method as a viable method for simulating gases. Though this textbook was, and still is, quite famous for its groundbreaking information, it was perceived by some as containing excessive information and in some cases the theory relevant specifically to DSMC was hidden amongst larger derivations. In 2013, Bird authored another textbook which aims to narrow the discussion specifically to information related to the implementation of the DSMC method, with little derivation, and a variety of test cases run to validate the method.

In conjunction with the time that Bird developed the DSMC method, there were other theoretical developments in fields relating to DSMC that have furthered the capabilities and strengethened the reputation of the method. One of the most prominent ways the DSMC method has improved is in the area of molecular and collision modeling. These developments have given the DSMC method arguably more ability over approximating the standard Boltzmann transport equation (BTE) since the BTE is unable to account for chemical reactions. Also, other investigations have bolstered the reputability of the method. In 1992, Wagner proved that the DSMC method converges to a solution to the Boltzmann equation in a suitable limit [20], and since then many other convergence studies have followed ([21], [22], [23]).

Several DSMC codes have arisen over the years, with the rate of appearance of new codes seemingly increasing. Naturally, Bird developed one of the first codes called DS1V, which is a general program for steady or unsteady one-dimensional flows. These were later followed by the two- and three- dimensional versions called DS2V and DS3V, respectively [24]. These programs can be freely downloaded at Bird's personal website [25], however access to the source codes has recently been restricted. These programs have primarily served as tools for validation cases utilized

by newer, developing DSMC codes. Another code developed shortly after the publication of Bird's 1994 textbook is called SMILE (Statistical Modeling In Low-density Environment), which came from the Siberian Division of the Russian Academy of Sciences in 1998 [26]. This code was utilized by many Russian and European space vehicle projects including analysis of the high-altitude stages of reentry of the Mir Space Station [27]. Other early DSMC codes came from the U.S. government or U.S. government contractors. The two main early codes were Sandia National Laboratories' Icarus and NASA's DAC (DSMC Analysis Code). Icarus is a 2D DSMC code optimized for parallel computing and capable of complex chemistry and ion transport modeling [28], while DAC is a general-purpose 3D DSMC code compatible with Cartesian grids and validated using real data from complex geometries such as the Space Shuttle [29].

All of the aforemetnioned codes were written in the Fortran programming language, but more recently, other DSMC codes have been developed in C++ to take advantage of the C++ programming language's object-oriented and data-oriented paradigms. One of these main codes is SPARTA (Stochastic PArallel Rarefied-gas Time-accurate Analyzer), which was developed out of Sandia National Laboratories and branched from their previous code Icaraus [30]. SPARTA is capable of 2D, 3D, and axisymmetric flows, and has proven effective in a wide variety of test cases related to hypersonic flows involving chemical reactions and ionizations on parallel computing platforms [31]. Similarly, NASA's Multiphysics Algorithm with Particles (MAP) was also written in C++ and branched from its Fortran-written predecessor, DAC. MAP has proven successful in simulating basic test cases such as flow over a sphere and more complex test cases such as flow around the Orion Crew Module [32]. The last popular DSMC code written in object-oriented C++ is an open source development code called dsmcFoam+ (formerly known just as dsmcFoam) which uses the popular CFD open source framework known as OpenFOAM [33]. This code is compatible

with parallel computers and has been successfully validated over standard test cases against other trustworthy DSMC codes such as SPARTA and DAC [34].

Several university research groups have developed and adopted their own versions of DSMC codes that are amongst the commercial code caliber. Cornell University created the code MONACO in collaboration with Professor Ian Boyd's Nonequilibrium Gas and Plasma Dynamics Laboratory (NGPDL) at the University of Michigan [35]. MONACO is a 2D, 3D, and axisymmetric DSMC code which has served as a backbone for commercial projects and codes. The University of Minnesota's Molecular Gas Dynamics Simulation (MGDS) Laboratory led by Prof. Thomas Schwartzentruber has created a 3D massively parallel DSMC code called the Molecular Gas Dynamics Simulator (MGDS) code, named after the laboratory that created it. The MGDS lab has published an article on the detailed implementation of parallelizing their DSMC code [36] and have published several articles on coupling the DSMC method to continuum fluid solvers [37] [38]. One of the most recent DSMC code developments comes from the Indian Institute of Technology Kanpur where the Non-equilibrium Flow Simulation Lab (NFSL) led by Prof. Rakesh Kumar developed the Nonequilibrium Flow Solver (NFS) 3D, multi-species, parallel DSMC code [39].

In 2008, Brian Saponas from Cal Poly [40] developed a 2D DSMC code on unstructured Cartesian grids. The code demonstrated successful simulations of supersonic flat plate and turning flow test cases. This is the only known DSMC code to come out of Cal Poly but due to the non-modular design of the code, which limits its use to specific classes of problems, and little documentation on the structure of the code itself, it was not further developed by future students.

1.3 Thesis Goals

The goal of the thesis project itself was to develop a DSMC code capable of simulating rarefied gases on a Cartesian grid and match validation test cases provided by Bird. The idea is that this code would lay the foundation for a large, multiphysics, multiscale code that future Cal Poly students can expand upon and use as a research tool for various projects. In order for the code to accomplish this, several key requirements were established:

1. Code is to be well-documented such that future programmers understand the work flow and implementation details
2. Code is to be written in a modular format such that small changes and future additions can be easily implemented without the code failing
3. Code must run primarily on an input file to not only mimic the process of commercial and research physics simulation codes, but also limit potentially dangerous user contact with the source code itself.
4. Code must demonstrate the ability to simulate physical processes within reasonable accuracy with the knowledge that more complex models will be added in future theses

The goal of this thesis report is to describe the general details of the DSMC method and outline the development, capabilities, and limitations of a new 3D DSMC simulation code, named SINATRA (SImulatioN of rArefied gases in the upper aTmospheRe And potentially plasma plumes), written in object-oriented C++. This thesis will start by fully describing the details of the DSMC method and the fundamental principles from which it was developed. Next, specific procedures required to simulate a flow field at the molecular level will be summarized. Specific computational considerations will then be discussed which include the benefits of object-oriented C++ and how that influenced the program structure and performance. Next, details on

how the DSMC method was implemented in SINATRA will be outlined. This thesis will then discuss specific test cases the code successfully completed, demonstrating its viability for future growth. The final chapter will discuss a path for the future development of the code.

Since future thesis projects aiming to further develop the SINATRA DSMC code have been planned, it is the intention of this report to be very detailed and at certain points, overly detailed, about the inner structure and workings of the code. The idea is that this report should be used as a reference for future developers. However, in the areas where code specifics fall short in this report, the User's Manual can be referenced for further details.

CHAPTER 2

DSMC: Theory and Procedures

2.1 Overview

The DSMC method is primarily founded on concepts from fields of statistical thermodynamics, kinetic theory of gases, and basic quantum mechanics. This includes fundamental characteristics of the molecular behavior of gases and how macroscopic properties can be formulated from microscopic behavior. This section attempts to introduce the relevant concepts used in the DSMC method from the aforementioned topics, as well as introduce the procedures required to implement the method in a computational code. Where appropriate, this section will also clarify which aspects of the procedures are incorporated into SINATRA and where those implementation details lie in Chapter 3.

2.2 Kinetic Theory

As mentioned in the Introduction, the DSMC method looks at the behavior of individual molecules and views a gas flow as the sum of those molecules' behavior. This description of a gas is commonly referred to as the kinetic theory of gases, or "kinetic theory" for short. In order to simulate flows through the interaction of individual molecules a few key concepts need to be understood and specified: an understanding of how the particles' properties are distributed throughout a flow field, the manner in which particles interact with one another, and how the particles' properties change due to interactions with each other.

2.2.1 Velocity Distribution Functions

Discussion of the kinetic theory would be incomplete without a summary of the velocity distribution functions which describe how each particles velocity is distributed in a particular flow field. Although focus of this section will be centered around a particle's velocity, these concepts can be directly extended to the distribution of other properties, such as energy, as well.

First define $\underset{\sim}{\boldsymbol{c}}$ as a molecule's velocity with components u, v, and w, corresponding to the x, y, and z Cartesian directions, respectively. Then, the velocity distribution function $f(\underset{\sim}{\boldsymbol{c}})$ is defined by:

$$\mathrm{d}N = Nf(\underset{\sim}{\boldsymbol{c}})\mathrm{d}u\mathrm{d}v\mathrm{d}w \tag{2.1}$$

where $\mathrm{d}N$ is the number of molecules in the sample with velocity components u to $u + \mathrm{d}u$, v to $v + \mathrm{d}v$, and w to $w + \mathrm{d}w$. Since, $\mathrm{d}u\mathrm{d}v\mathrm{d}w$ can be identified as the differential volume element $\mathrm{d}\underset{\sim}{\boldsymbol{c}}$, Equation 2.1 can be written as in the more general form:

$$\mathrm{d}N = Nf(\underset{\sim}{\boldsymbol{c}})\mathrm{d}\underset{\sim}{\boldsymbol{c}} \tag{2.2}$$

For the sake of conciseness, the functional relation is usually omitted so that $f(\underset{\sim}{\boldsymbol{c}})$ is simply written as f.

Since every molecule in the gas is represented by a point in velocity space, the distribution function is a normalized function such that its integration over the entire velocity space yields unity, as shown below in Figure 2.3. This is an important characteristic to remember for all probability distribution functions used in the DSMC method.

$$\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f \mathrm{d}u\mathrm{d}v\mathrm{d}w = \int_{-\infty}^{\infty} f \mathrm{d}\underset{\sim}{\boldsymbol{c}} = 1 \tag{2.3}$$

These probability distribution functions can be extended to multiple species gas mixtures, however these equations are not summarized in this work. For details on those equations, see Section 3.1 in [11], where all the equations in this section were sourced.

These functions are statistical descriptions in terms of probability distributions which must be employed since it is unreasonable to list the position, velocity, and internal state of every molecule at a particular instant, which is what would be required for analytical descriptions.

2.2.2 Binary Collisions

A key aspect of the DSMC method that must be defined is how particles interact with one another. This is the next logical step after the particles' properties are given using the probability distribution function discussed previously. It has been found that intermolecular collisions in dilute gases are extremely likely to involve just two molecules, as shown below in Figure 2.1. These interactions are called "binary collisions" and are the focus of this section. In fact, this entire thesis only accounts for binary collisions, in theory and in code implementation.

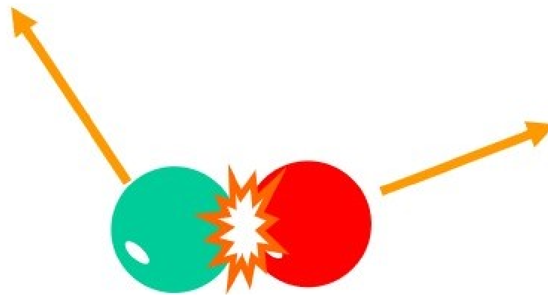


Figure 2.1 – Depiction of a binary collision between two particles [6].

In order to predict how molecules' properties change after collisions, the following properties must be known: the molecule's mass, pre-collision relative velocity between molecules, $\underset{\sim}{\boldsymbol{c}}_r$, and impact parameters such as the distance of closest approach, b, and the angle between the collision plane and some reference plane, ϵ. Note that these variables are similar to those involved in the Boltzmann Transport Equation shown in Equation 1.3. Given that knowledge of these parameters is often unknown or the mathematics behind them are intractable, molecular models are often used to reproduce the effect of those particular features, but do not explicitly incorporate those features in the model. These types of models are known as "phenomenological models", and the DSMC method relies heavily on these for collision modeling. Section 2.2.3 summarizes the models used in SINATRA, though many more advanced models exist today.

2.2.3 Molecular Models

As mentioned previously, phenomenological molecular models aim to reproduce the observed properties of a real gas. Instead of using the impact parameters necessary in the Boltzmann equation, the DSMC method utilizes different collision cross sections to achieve the desired effect of realistic transport properties. Thus, the primary difference between "molecular models" in DSMC is how the collision cross section is determined and, in some cases, how the molecules' velocities change after impact.

The two primary models used in SINATRA are the Hard Sphere (HS) and Variable Hard Sphere (VHS) models.

The Hard Sphere model is the original model used in the DSMC method, and naturally, is the most simple model to implement. Here, particles are modeled similarly to billiard balls in that their diameters remain constant regardless of environmental conditions. Although this is simplest model to envision, it has failed to produce reasonable transport property results for flows with velocity and temperature gradients.

The Variable Hard Sphere model attempts to correct the shortfalls of the HS model by allowing the diameter (and the cross section) to change based on translational energy, which effectively allows transport properties to exhibit a realistic temperature dependence. This is the preferred method for a simple DSMC code, as it produces realistic results through simple procedures.

Both the HS and VHS models employ random scattering laws which mean that the post collision velocity directions are independent of pre-collision properties. More recently, other molecular models have been developed that better simulate molecular interactions (such as temperature dependence and post-collision scattering), but these models are out of the scope of this thesis.

2.3 Basic Quantum Theory

The DSMC method uses aspects from elementary quantum mechanics when simulating molecular energies to capture the true microscopic description of gases. Certainly, a full description of quantum mechanics is not within the scope of this thesis, but there are certain aspects to quantum theory that must be understood to effectively use the DSMC method. These aspects primarily involve the quantities and terminology associated with molecular energies. The explanation of these topics are summarized primarily from [8].

A molecule's total energy, ε_{tot}, is quantified as the sum of its translational energy, ε_{trans}, rotational energy, ε_{rot}, vibrational energy, ε_{vib}, and electronic energy, ε_{el}:

$$\varepsilon_{tot} = \varepsilon_{trans} + \varepsilon_{rot} + \varepsilon_{vib} + \varepsilon_{el} \tag{2.4}$$

Each of these components of the total energy are said to be a "mode" of molecular energy. Additionally, each mode of energy contributes a unique amount of "degrees of freedom" (or "thermal degrees of freedom") to the molecule.

The total energy is often segregated into two components: the translational energy and internal energy, ε_{int}, where:

$$\varepsilon_{int} = \varepsilon_{rot} + \varepsilon_{vib} + \varepsilon_{el} \tag{2.5}$$

Thus,

$$\varepsilon_{tot} = \varepsilon_{trans} + \varepsilon_{int} \tag{2.6}$$

Translational energy comes directly from the translational kinetic energy of the center of mass of the molecule. Since there are three components of velocity (in the x-, $y-$, and $z-$ directions), the molecule has three thermal degrees of freedom in translation. Figure 2.2 shows a standard depiction of molecular translational motion that contributes to the translational energy of the molecule.

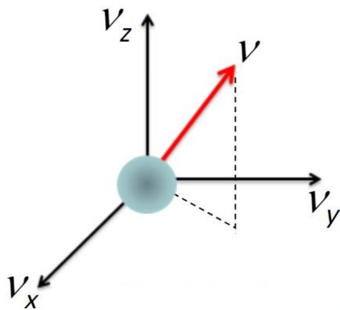


Figure 2.2 – Depiction of Translational Energy.

Rotational energy exists if the molecule is rotating about the three orthogonal axes in space. The source of this energy is the rotational kinetic energy associated with the molecule's rotational velocity and its moment of inertia. An illustration of rotational energy is shown below in Figure 2.3.

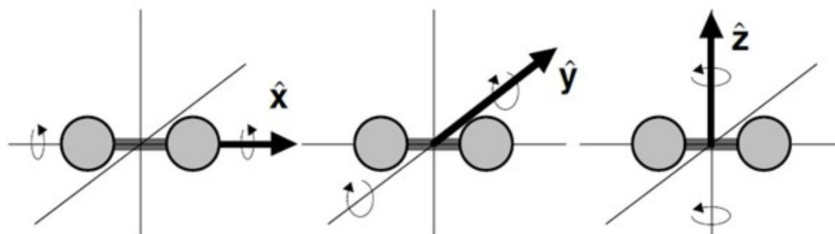


Figure 2.3 – Depiction of Rotational Energy [7].

It is worth noting that monotomic molecules do not have any rotational energy, so they effectively have zero degrees of freedom in rotation. Figure 2.3 shows a diatomic molecule which only significantly rotates about two axes, since the moment of inertia about the x-axis is negligible. Thus, most diatomic molecules only have two degrees of freedom in rotation. Also, polyatomic molecules generally have more than two degrees of freedom in rotation, but the exact number varies based on the complexity of the molecular structure.

Vibrational energy arises when atoms of a molecule vibrate with respect to an equilibrium location within the molecule. This is often represented by a spring as shown in the diatomic molecule of Figure 2.4.

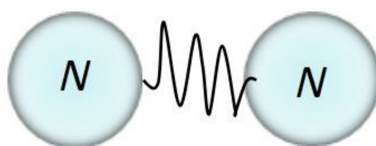


Figure 2.4 – Depiction of Vibrational Energy.

There are two sources of energy from vibration: kinetic energy from the linear motion of the atoms vibrating back and forth, and potential energy associated with the intermolecular force keeping the atoms together. Thus, there are two thermal degrees of freedom in vibration for a simple diatomic molecule. Polyatomic molecules experience much more complex vibrational modes since often time their structure is

not linear, so vibration can occur along multiple axes. This can cause a polyatomic molecule to have a large number of degrees of freedom.

Electronic energy comes from the motion of electrons as they orbit the nucleus of each atom. Further details of electronic energy are out of the scope of this thesis as electronic energy in ignored in the DSMC code.

Energy levels of a molecule are quantized [8], so the exact quantities of energy can only exist at specific discrete values for each energy mode.

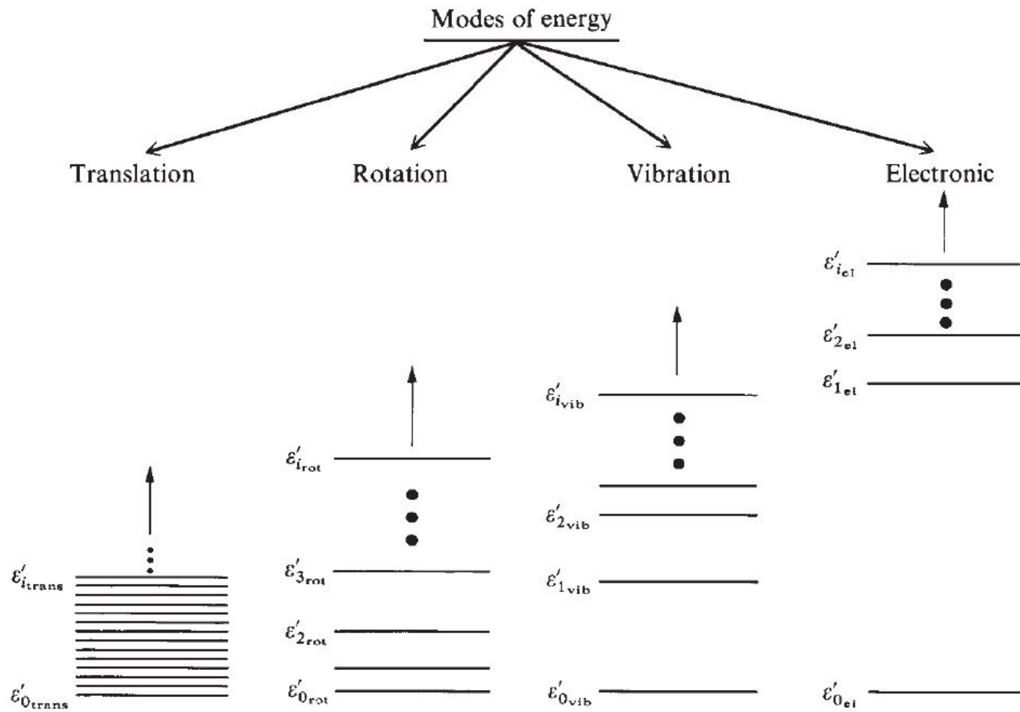


Figure 2.5 – Schematic of energy levels for different energy modes [8].

To simplify the model when implementing it in Bird's DSMC method, translational and rotational energy are assumed continuous while vibrational energy follows the "simple harmonic model" which assumes that the energy levels are evenly spaced out. Since the possible translational and rotational energy levels are closely spaced, it is a safe assumption to model them as continuous. However, since the vibrational

energy levels are spaced further apart, the simple harmonic model is a better assumption.

2.4 Macroscopic Properties

The beauty behind molecular descriptions of gases is the ability to extract macroscopic properties from microscopic behavior. These macroscopic properties are tangible and are what can be measured easily in a laboratory setting. The primary macroscopic properties concerned here are density, velocity, temperature, pressure. From these, secondary quantities such as the viscous stress tensor and the heat flux vector can be computed (although there are various others than can be found as well).

One of the key strategies to link the microscopic properties to the macroscopic properties lies in the relation between constants. The most important equation, formulated by Ludwig Boltzmann, links entropy, S, with the probability of molecules being at certain microstates, W, in the equation:

$$S = k \log W \tag{2.7}$$

where k is the Boltzmann constant quantified by $k = 1.3806488x10^{-23} J/K$. This equation is said to be the "bridge between classical thermodynamics (represented by S) and statistical thermodynamics (represented by W)" [8]. From the Boltzmann constant, other microscopic-macroscopic physical constant values can be related.

One important physical constant in the analysis of gas flows is the universal (or molar) gas constant, $\Re$, given by:

$$\Re = N_A k \tag{2.8}$$

where N_A is Avogadro's number which is the number of atoms or molecules in 1 mol of an atom or molecule, respectively, and is quantified by $N_A = 6.02214129x10^{23}/mol$. This produces the universal gas constant equating to $\Re = 8.3144621 J/mol/K$.

In some cases, it is more convenient to find the specific (or ordinary) gas constant unique to each gas species. The specific gas constant can be found by taking the ratio of the Boltzmann constant and the molecular mass, m of the chemical species in question:

$$R = \frac{k}{m} \tag{2.9}$$

or when looking at the bulk gas mixture, the specific gas constant can also be found by:

$$R = \frac{\Re}{m} \tag{2.10}$$

where R is unique to each species and the variables $\Re$ and m were defined previously.

And finally, the molecular mass of each species can be found by dividing the mass of the overall gas, M, by Avogadro's number:

$$m = \frac{M}{N} \tag{2.11}$$

which rounds up the relations between physical constants.

These relations appear most notably in the form of the ideal gas equation of state, which can be written using both microscopic and macroscopic constants:

$$p = nkT = \rho RT \tag{2.12}$$

The density can be found by:

$$\rho = nm \tag{2.13}$$

where n is the "number density", defined as the number of molecules of a given species in a unit volume.

Continuum fluid mechanics is concerned with a fluid's average (or stream) velocity. At the molecular level, every individual molecule has its own velocity, $\underset{\sim}{\boldsymbol{c}}$, and the mean molecular velocity $\overline{\underset{\sim}{\boldsymbol{c}}}$ defines the stream velocity, which is denoted by $\underset{\sim}{\boldsymbol{c_0}}$:

$$\underset{\sim}{\boldsymbol{c_0}} = \overline{\underset{\sim}{\boldsymbol{c}}} \tag{2.14}$$

The velocity of an individual molecule relative to the stream velocity of the entire gas is called the "thermal" or "peculiar" velocity and is denoted by $\underset{\sim}{\boldsymbol{c'}}$:

$$\underset{\sim}{\boldsymbol{c'}} = \underset{\sim}{\boldsymbol{c}} - \underset{\sim}{\boldsymbol{c_0}} \tag{2.15}$$

where $\underset{\sim}{\boldsymbol{c}}$ is the velocity of an individual particle. Thus every particle in a mixture has a different thermal velocity. The peculiar velocity is particularly important in the study of statistical thermodynamics as many macroscopic properties vary based on the magnitude and direction of this quantity.

Contrary to the continuum Navier-Stokes model which assumes that pressure is a scalar quantity, the pressure in the molecular model is a function of two directions and appears as a tensor quantity. The pressure tensor can be found by:

$$\underset{\sim}{\boldsymbol{p}} = \begin{pmatrix} \rho\overline{u'^2} & \rho\overline{u'v'} & \rho\overline{u'w'} \\ \rho\overline{v'u'} & \rho\overline{v'^2} & \rho\overline{v'w'} \\ \rho\overline{w'u'} & \rho\overline{w'v'} & \rho\overline{w'^2} \end{pmatrix} \tag{2.16}$$

or in indicial notation:

$$p_{ij} = \rho\overline{c_i'c_j'} \tag{2.17}$$

The scalar pressure is the mean of the three normal components (elements along the diagonal) of the pressure tensor. This is written as:

$$p = \frac{1}{3}\rho(\overline{u'^2} + \overline{v'^2} + \overline{w'^2}) = \frac{1}{3}\rho\overline{c'^2} \tag{2.18}$$

or in indicial notation:

$$p = \frac{1}{3}\rho\overline{u_i'u_i'} = \frac{1}{3}\rho p_{ii} \tag{2.19}$$

However, it is convenient to write this in terms of the translational temperature, T_{tr} in the form:

$$p = nkT_{tr} \tag{2.20}$$

The viscous stress tensor, τ_{ij}, is deduced from the pressure tensor in that it is the negative of the pressure tensor with the scalar pressure subtracted from the normal components. In indicial notation:

$$\tau_{ij} = -\big(\rho\overline{c_i'c_j'} - \delta_{ij}p\big) \tag{2.21}$$

The viscous stress tensor expanded in tensor form:

$$\underset{\sim}{\tau} = \begin{pmatrix} -(\rho\overline{u'u'} - p) & -\rho\overline{u'v'} & -\rho\overline{u'w'} \\ -\rho\overline{v'u'} & -(\rho\overline{v'v'} - p) & -\rho\overline{v'w'} \\ -\rho\overline{w'u'} & -\rho\overline{w'v'} & -(\rho\overline{w'w'} - p) \end{pmatrix} \tag{2.22}$$

A course in Viscous Flow must be taken or a Viscous Flow textbook must be thoroughly examined to understand the relations between the transport phenomena and the significance of the viscous stress tensor. A good source for such reading is found in [41].

2.5 Transport Properties

The Chapman-Enskog theory gives a solution to the Boltzmann equation for a small set of problems in which the distribution function, f, is perturbed by a small amount from the equilibrium Maxwellian form. The distribution function can be expressed in the form of a power series using the form:

$$f = f_0(1 + \Phi_1 + \Phi_2 + ...) \tag{2.23}$$

Chapman and Enskog provided solutions to the first-order form of Equation 2.23:

$$f = f_0(1 + \Phi_1) \tag{2.24}$$

The coefficient of viscosity for a VHS gas is given by Equation 2.25 below:

$$\mu = \frac{15(\pi m k)^{1/2}(4k/m)^{\omega - 1/2} T^{\omega}}{8\Gamma(9/2 - \omega)\sigma c_r^{2\omega - 1}} \tag{2.25}$$

The first approximation to the coefficient of heat conduction, K, is related to the coefficient of viscosity by:

$$K = \frac{15k\mu}{4m} \tag{2.26}$$

Thus, the Prandtl number in a monatomic gas is:

$$Pr = \frac{\mu c_p}{K} = \frac{2}{3} \tag{2.27}$$

The Chapman-Enskog theory also gives the diffusion coefficients for two species 1 and 2 as:

$$D_{12} = \frac{3\pi^{1/2}(2kT/m_r)^{\omega_{12}}}{8\Gamma(7/2-\omega_{12})n\sigma_{12}c_r^{2\omega_{12}-1}} \tag{2.28}$$

where the subscript 12 denotes the quantity between two different species, and m_r is the reduced mass calculated by Equation 2.29 below:

$$m_r = \frac{m_1 m_2}{m_1 + m_2} \tag{2.29}$$

which is a common value computed in collision modeling.

The coefficient of self-diffusion is:

$$D_{11} = \frac{3\pi^{1/2}(4kT/m)^{\omega}}{8\Gamma(7/2-\omega)n\sigma c_r^{2\omega-1}} \tag{2.30}$$

where the subscript 11 denotes the quantity for interactions between the same species.

Finally, the Schmidt number, Sc, is a dimensionless number defined as the ratio of momentum diffusivity and mass diffusivity [42], and is given by:

$$Sc = \frac{\mu}{\rho D_{11}} = \frac{5}{7-2\omega} \tag{2.31}$$

These results have been verified by Chapman and Cowling [43]. The results of these transport properties form the basis for the species parameters necessary to input into a DSMC simulation. These are summarized in property tables given by Bird in the Appendices of [11] and [44].

2.6 DSMC Basic Procedure

The basic outline of the DSMC procedure is given below in Figure 2.6. Every block in the flow chart involves some aspect of the theory described in this section.

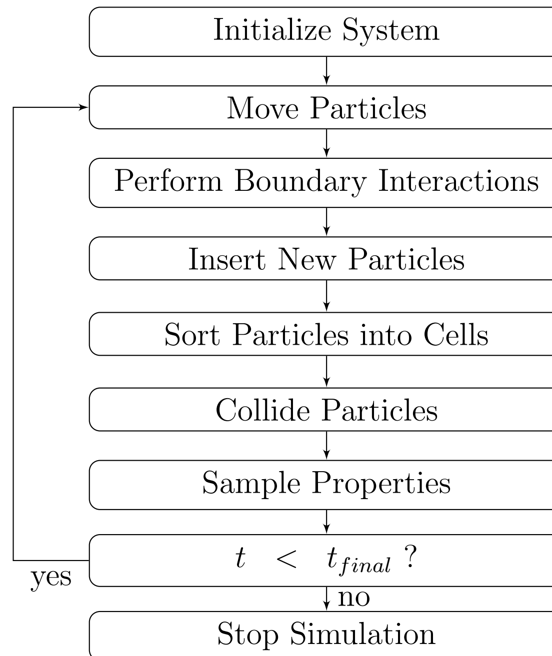


Figure 2.6 – Flow Chart of General DSMC Procedures

It is expected that all DSMC codes perform the basic procedure set forth by Figure 2.6. However, the implementation of these procedures varies from code to code, and there is a challenge associated with each step.

CHAPTER 3

SINATRA Code Structure

3.1 Object-Oriented Programming and C++

In a broad sense, computer programming is the act of communicating with a computer by giving it a set of instructions, or code, that tells it what to do. However, this can be accomplished in many different ways. A programming paradigm can be thought of as a school of thought on how to program on a computer. There are four main programming paradigms as discussed in [45].

1. Imperative paradigm
2. Functional paradigm
3. Logical paradigm
4. Object-oriented paradigm (OOP)

Although much can be said about each school of thought, discussion will be restricted to the object-oriented paradigm. The primary characteristics of OOP are encapsulation, inheritance, and polymorphism [46]. Encapsulation is the idea that since objects operate independently, they are encapsulated into modules which contain both local variables and methods. This allows for easy information hiding and organization. The idea of inheritance captures how objects can easily be created from others by inheriting data and functions from base classes. This allows for easy code reuse and extension without changing existing source code [47]. Finally, polymorphism is the concept that functions can take different forms depending on the situation in which they are used [48]. One of the primary benefits of OOP as stated

in [49] is its ability for "direct expression" which means that objects are natural metaphors for both physical objects and abstract entities. In essence, this allows the programmer to see data evolve conceptually as controlled by the program. Since the programming language is closely tied to the paradigm desired, SINATRA was developed in object-oriented C++ to take full advantage of all these benefits and effectively represent physical particles through physically realistic data structures.

A fundamental requirement, and ultimately benefit, of C++ is the use of pointers which allow for memory control of variables. Rather than explicitly specifying variable names when accessing data, the memory address is pointed to. Since DSMC codes can create thousands, if not millions, of data structures pointers are the preferred method for accessing and operating on data. Figure 3.1 below shows a basic conceptual diagram of how pointers work. Note that the column on the left represents the memory address and the column on the right represents the variable that is stored. The pointer variable, in this case `Yptr`, stores the address, but when the pointer is dereferenced, in this case by calling `*Yptr`, the value at the address is accessed.

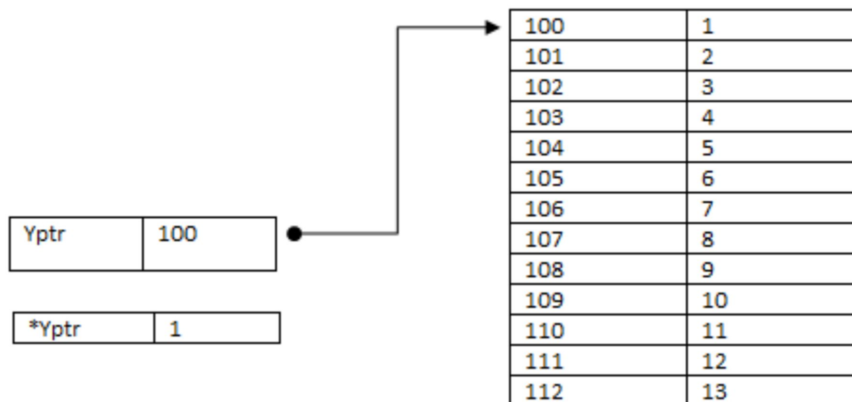


Figure 3.1 – Diagram of pointers [9].

Although this was only a basic example, this concept is extended to the data structures involved in SINATRA and provides the foundation for the DSMC structure

and implementation.

3.2 Cell and Particle Data Structures

The code structure of SINATRA primarily revolves around two key concepts:

- The storage of and interactions between cell and particle data structures.
- The interaction between classes that operate on one another to drive the flow of the code.

With regards to the first concept, the data structures store relevant information about the cell and particle objects and bring physical meaning to the data they store. In order to fully understand this, a detailed description of the computational grid that SINATRA is intended for is given. Next, the details of the particle structures and information they hold are described. Finally, the method for which the two data structures are linked is fully described.

3.2.1 Cell Structure

The computational grid is the domain for which simulations take place. This grid captures reality by taking the physical space in which the activity in question occurs and converting it into computer space where simulation activity takes place. It is common practice in computational mechanics to discretize the domain into a mesh. While traditional CFD programs use the mesh to compute the properties of the flow field and depend on numerical analysis proofs dictating the size and location of the cells, the DSMC method primarily uses the discretized domain to track particles for collision calculations and macroscopic property extraction.

SINATRA is intended to only be used for "Cartesian grids" which are a a special case of regular grids where the elements are unit cubes and the sides are aligned

with the axes of the Cartesian coordinate system [50]. The primary reason for using this type of mesh in SINATRA is to take advantage of the "Octree Data Structure" which is a hierarchical data structure which utilizes a recursive decomposition of space [51]. From a data structure perspective, the domain is represented by one root data structure which produces eight new data structures during the first recursive step. These new data structures are the "children" of the root data structure. At each successive recursive step, any of the children can produce eight new children. This can be visualized with an octree tree diagram as shown below in Figure 3.2:

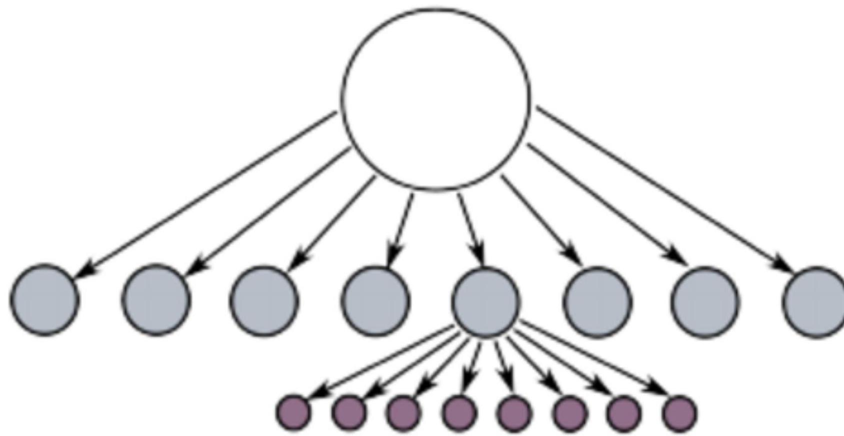


Figure 3.2 – Tree diagram of octree data structure [10].

The octree structure can also be visualized with blocks, where the entire domain starts off as one block then each coordinate side is divided in half, creating 8 new blocks per split block at each recursive step. From the perspective of the computational domain, a single cell is evenly divided into 8 subcells by cutting the cell in half in each Cartesian direction. This occurs recursively until the desired amount of refinement is obtained. An illustration of how a cell is cut can be seen below in Figure 3.3.

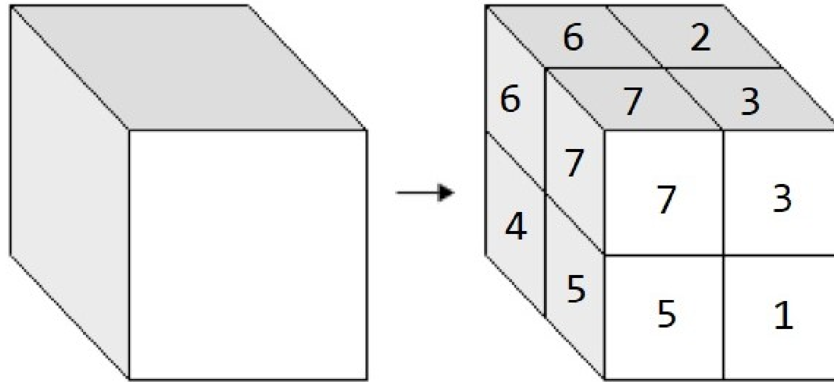


Figure 3.3 – Cell cut in Octree fashion.

Notice that in the refined cell, the subcells are numbered up to 7 (note that 0 is not shown since it is in the bottom back corner). This type of numbering scheme is always used in SINATRA to label which of the children are being referenced, where the possible labels range from 0 to 7. Table 3.1 shows which number corresponds to which relative subcell in the octree data structure.

Table 3.1 – Relative positions of children cells.

Child Cell Label	Relative Location
0	low x, low y, low z
1	high x, low y, low z
2	low x, high y, low z
3	high x, high y, low z
4	low x, low y, high z
5	high x, low y, high z
6	low x, high y, high z
7	low x, high y, high z

The idea is that it is possible to cut a cube into 8 "child" cubes, and repeatedly cut one or many of these children cubes into 8 more children cubes each.

Using the Octree terminology, the original, un-cut cube is called the "root node", or in the case of using the cube as the domain it is called the "root cell". When cutting this cube, or any cube into eighths, it is called a "parent cell", and the

subcells produced from these cuts are called "child cells". These children cells can also be cut further, so the children cells that are cut are also considered parent cells. This process is repeated until the desired refinement (or cutting) is achieved. The smallest cells, which are not cut any further, are called "leaf cells", or "leaf nodes" in general octree terminology. It should be made clear that not all cells need to be cut to the same refinement, so the number of cuts in any cell can vary from cell to cell as shown below in Figure 3.4.

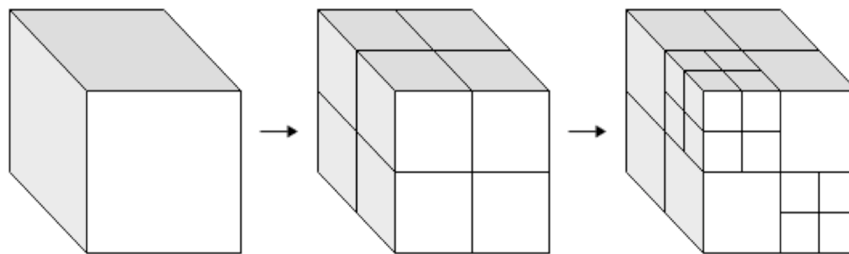


Figure 3.4 – Varying refinement from cell to cell.

The way the octree structure is implemented is by utilizing two different `structs` in C++, which are similar to classes except that they do no store member functions. One struct is `ParentCell` which is used to represent the parent cells of the grid and the other is called `ChildCell` which is used to represent the child cells of the grid. As described previously, these structs store relevant information, in the form of variables, about the cell in question.

There is a supplementary struct called `ChildInfo` that stores extra information relating directly to every individual child cell. There is one `ChildInfo` struct for every `ChildCell` struct. The reason this information is separated into two different structs is given below after the details of the structs are described.

Table 3.2 below shows the members (or variables) of the `ChildCell` struct.

Table 3.2 – Description of child cell variables in `ChildCell` struct.

Variable	Description
`id`	Child ID number
`indexOfParentCell`	Index of parent cell in `parentCells` vector
`indexOfThisCell`	Index of own cell in `childCells` vector
`neighbors[6]`	Array of neighboring cell indices in `childCell` vector
`low_corner[3]`	Array of xyz coordinates of minimum corner
`high_corner[3]`	Array of xyz coordinates of maximum corner
`maxCrossSpeed`	Maximum value of σc_r for collision computation

The ID number is unique to every child cell and identifies one specific child cell in the domain. In SINATRA's current serially programmed iteration, the ID number also happens to be one integer value greater than the `indexOfThisCell` variable. Although this is redundant information in SINATRA's current version as all data is accessed from the same processor, this is necessary information when the code becomes parallelized. The `neighbors` integer array stores the indices of the child cell's direct neighbors that its entire face is in contact with. Note that these neighbors do not necessarily have the same parent cell. Also, if a child cell happens to be at the edge of the domain such that one, or more, of its faces is in contact with the boundary, the `neighbors` array stores an integer corresponding to the boundary type. Each index of `neighbors` corresponds to one face in a particular coordinate direction as specified by Table 3.3 below.

Table 3.3 – Relative location of each index in `neighbors` array.

Index	Face Location
0	xlow
1	xhigh
2	ylow
3	yhigh
4	zlow
5	zhigh

The indexing described in Table 3.3 is held consistent throughout SINATRA. The

`low_corner` and `high_corner` type double array stores the x, y, and z coordinates of the lowest and highest corners, respectively, of the child cell. This indexing is also consistent with the x, y, and z indexing used throughout SINATRA. For the sake of clarity, this indexing is specified below in Table 3.4.

Table 3.4 – Index key for each coordinate direction.

Index	Coord.
0	x
1	y
2	z

Table 3.5 below showns the members of the `ChildInfo` struct. This struct holds extra information relavent to the children cells which are pertanent to the DSMC simulation. This information is most useful during the child-particle linking discussed in Section 3.2.3.

Table 3.5 – Description of child cell information variables in `ChildInfo` struct.

Variable	Description
`numParticles`	The number of particles in the cell (leaf cells only)
`firstParticle`	Index in `particleArray` of the first particle in this cell
`volume`	Volume of cell
`isLeaf`	Flag if the child cell is a leaf cell

The Future Work chapter (Chapter 7) describes changes that should be made to the code, one of which is to store additional relevant information about the child cell, such as certain sampled properties. These added properties should be stored in the `ChildInfo` struct. Thus, it is expected that this struct will increase in size.

Table 3.6 – Description of parent cell variables in ParentCell struct.

Variable	Description
id	Parent ID number
indexOfParentCell	Index of this cell's parent in parentCells array
indexOfFirstChild	Index of first child in childCells array
level	Level in hierarchical grid
IsGrandparent	Flag if the parent is also a grandparent
low_corner[3]	Array of xyz coordinates of minimum corner
high_corner[3]	Array of xyz coordinates of maximum corner

These structs are linked together in STL vectors of their respective types. There is an STL vector called childCells and an STL vector called parentCells which store a list of all the child and parent cells, respectively. This allows for direct access to any cell in the domain. There is also an STL vector called childInfo (note the case-sensitivity and the distinct difference to its type) that stores all the ChildInfo structs in one continuous memory. There is a one-to-one correspondence of each child cell and the structure of each child info data structure. This was the primary reason for splitting the child cells' data between two different structs. Since the STL vectors store data in continuous blocks of memory, it may be beneficial to split up this memory to alleviate excessively large memory chunks.

An illustration of how the STL vectors link the structs together can be seen below in Figures 3.5, 3.6, and 3.7.

childCells[0]	**childCells[1]**		**childCells[n-1]**
- id	- id		- id
- indexOfParentCell	- indexOfParentCell		- indexOfParentCell
- indexOfThisCell	- indexOfThisCell		- indexOfThisCell
- neighbors[6]	- neighbors[6]	...	- neighbors[6]
- low_corner[3]	- low_corner[3]		- low_corner[3]
- high_corner[3]	- high_corner[3]		- high_corner[3]
- maxCrossSpeed	- maxCrossSpeed		- maxCrossSpeed

Figure 3.5 – Illustration of STL vector, called `childCells`, containing n `ChildCell` structs.

childInfo[0]	**childInfo[1]**		**childInfo[n-1]**
- numParticles	- numParticles		- numParticles
- firstParticle	- firstParticle	...	- firstParticle
- volume	- volume		- volume
- isLeaf	- isLeaf		- isLeaf

Figure 3.6 – Illustration of STL vector, called `childInfo`, containing n `ChildInfo` structs.

parentCells[0]	**parentCells[1]**		**parentCells[m-1]**
- id	- id		- id
- indexOfParentCell	- indexOfParentCell		- indexOfParentCell
- indexOfFirstChild	- indexOfFirstChild		- indexOfFirstChild
- level	- level	···	- level
- IsGrandparent	- IsGrandparent		- IsGrandparent
- low_corner[3]	- low_corner[3]		- low_corner[3]
- high_corner[3]	- high_corner[3]		- high_corner[3]

Figure 3.7 – Illustration of STL vector, called `parentCells`, containing m `ParentCell` structs.

In SINATRA's current serially programmed iteration, the ID number of each parent cell also happens to be its own index in its vector. Although this is redundant information when the code is written in serial, this distinction becomes important if the code is written in parallel and the continuous chunks of memory are split amongst processors.

The C++ code for how the structs and vectors are declared in SINATRA is shown in Appendix C.1.

3.2.1.1 Accessing Children from Parents

In SINATRA, the way to access each child cell from the parent cell is to take advantage of the variable `indexOfFirstChild`. Since the rest of the children are stored successively in the `childCells` vector in the order given above in Table 3.1, to access any of the other children the programmer must only add the correct value to the index of the first child. A diagram of this procedure is shown below in Figure 3.8.

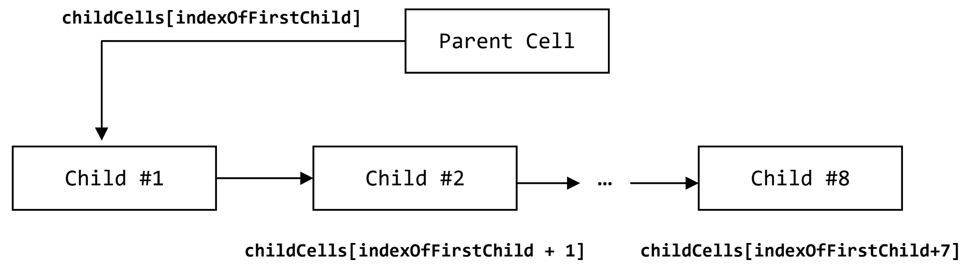


Figure 3.8 – Diagram of Parent-Child relationship.

3.2.1.2 Traversing the Octree Structure

Since only the `ParentCell` structures store children, for multiple levels of refinement, it is often the case where children cells also have children, and thus are parent cells as well. Functions were created to directly find the equivalence between children and parent cells. These functions are called `findParentToMatchChild()` which returns the parent that matches any child cell as long as the child cell is not a leaf, and `findChildToMatchParent()` which returns the child cell that matches any parent as long as it is not the root cell. Thus, to traverse the entire octree downwards and find the leaf cells from the root cell:

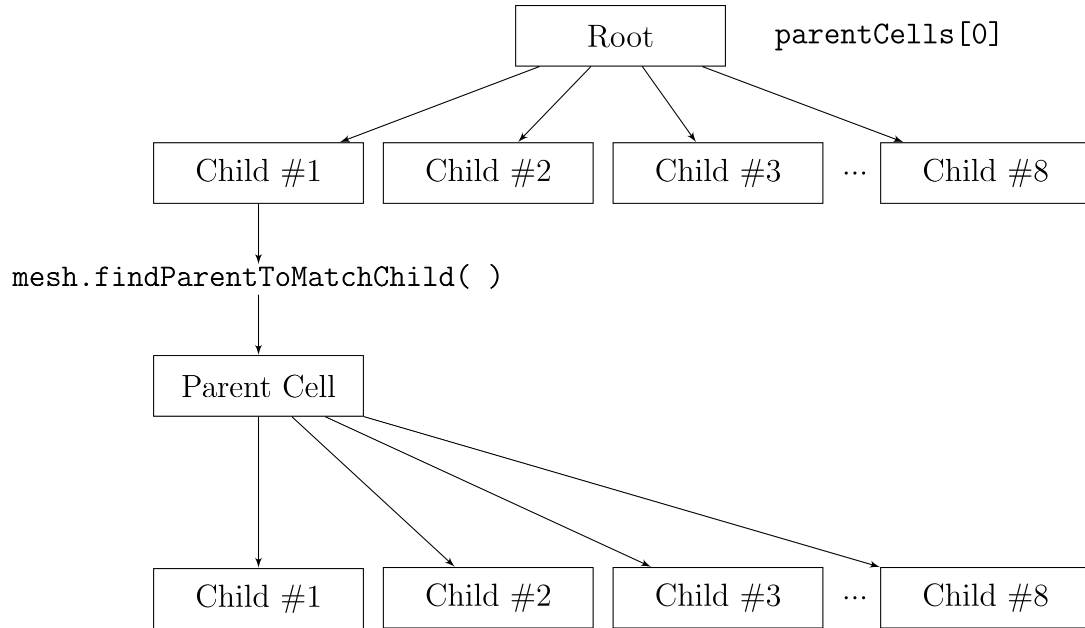


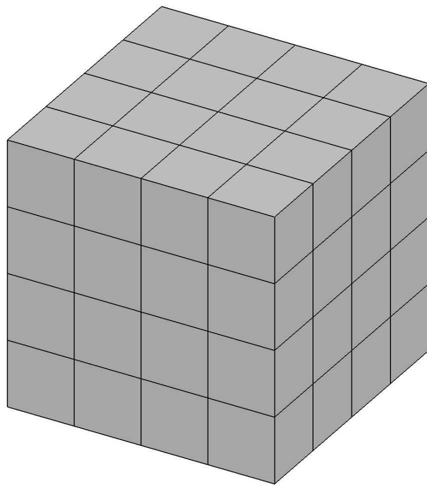
Figure 3.9 – Traversing octree data structure from root to leaf cell.

3.2.1.3 Computational Domain

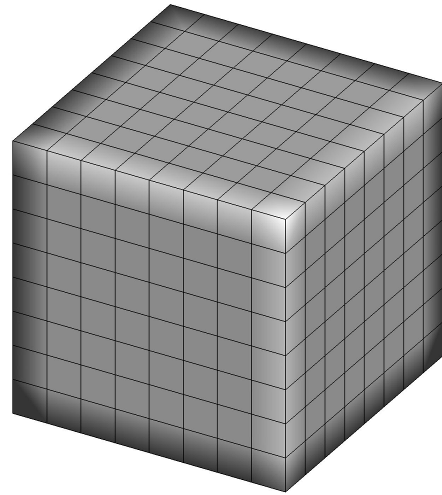
It was the initial, and is now the future, goal of SINATRA to use Cart3D as a Cartesian mesh generator to run DSMC simulations on. Unfortunately, due to licensing issues early on, utilization of Cart3D was not possible and the scope changed to use different meshes. In an effort to focus more on the DSMC aspects of the code rather than the meshing aspects (which are potentially full theses in themselves), the decision was made to develop the code on simple, home-grown meshes. The assumption was that through C++ libraries provided by Cart3D, data from Cart3D could be converted to the typical finite-element format which lists the coordinate locations of every node followed by a connectivity list that defines which nodes form an individual cell. So jumping straight to generating Cartesian meshes in the finite element format, a separate C++ function was developed to create simple uniform, Cartesian grids which refine each cell once on each successive refinement, following the octree structure. This function, called FEMeshGenerator() can be seen in Appendix A. Using

this function, the generated meshes (and thus the most used meshes for all SINATRA simulations) have 8, 64, 512, 4096, 32768, etc. cells.

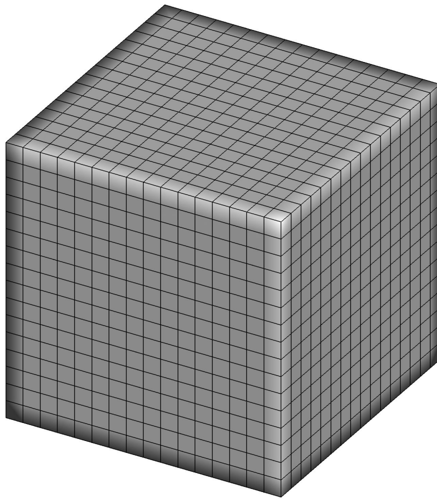
Images of the most common levels of refinement can be seen below in Figure 3.10.



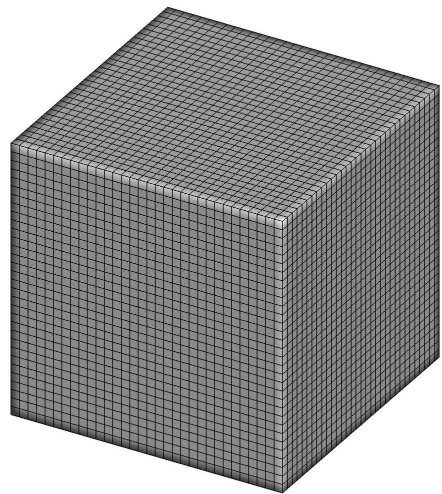
(a) 64 Cells



(b) 512 Cells



(c) 4096 Cells



(d) 32768 Cells

Figure 3.10 – Most commonly used grids for SINATRA.

It is worth noting that the mesh generating function is not integrated within the SINATRA code. It is a separate function that generates a text file of the nodes and connectivity list. The goal of SINATRA is to be able to read in separate mesh files

and run simulations on those grids. It is also worth noting that SINATRA is also able to run on non-uniform grids. However, since the function only generates uniform grids, non-uniform grids must be created by hand which are quite tedious, especially for very refined grids.

3.2.2 Particle Structure

Similar to the data structures that represent the cells in the computational domain, there is a struct called `OneParticle` that represents a single simulation particle. The `OneParticle` struct holds information related to a particle's position, velocity, and energy, as well as information identifying the type of particle. Table 3.7 below gives a summary of the variables stored in the `OneParticle` data structure.

Table 3.7 – Description of particle structure variables.

Variable	Description
`id`	Particle ID number
`indexOfSpecies`	Index of species in `speciesArray`
`indexOfChildCell`	Index of child (leaf) cell in `childCells` vector
`indexOfNextParticle`	Index of next particle in `particleArray` for linked list
`position[3]`	Array of xyz coordinates of the particle's location
`velocity[3]`	Array of the particle's uvw velocity components
`rotationalEnergy`	Magnitude of the particle's rotational energy
`vibrationalEnergy`	Magnitude of the particle's vibrational energy
`dtremain`	Fraction of time step remaining during BC interaction

All of the `OneParticle` structs, and ultimately all particles involved in the DSMC simulation, are stored in one continuous block of memory in the form of an STL vector called `particleArray`. In SINATRA's current state, every particle ID is offset from its index in `particleArray` by one. This however, will not always be true in future iterations of the code when it becomes parallelized, as the block of memory will be split amongst processors.

Discussion of these particle data structures are incomplete without a description of the `Species` struct, which describes a specie (or type of particle/chemical) in the simulation. All particles adopt a specie identity and inherit certain properties from that specie. All particles of the same specie, have these properties in common. Table 3.8 below gives a summary of the variables stored in the `Species` struct.

Table 3.8 – Description of species structure variables.

Variable	Description
`speciesNum`	Species number (similar to ID number)
`speciesFrac`	Number fraction of species for the entire mixture
`refTemp`	Reference temperature for the properties given
`molDiameter`	Molecular diameter of the species at the ref. temperature
`mass`	Mass of species
`rot_dofs`	The number of rotational energy degrees of freedom
`vib_dofs`	The number of vibrational energy degrees of freedom
`rot_temp`	Characteristic temperature for rotational energy
`vib_temp`	Characteristic temperature for vibrational energy
`omega`	Viscosity index
`viscCoeff`	Target coefficient of viscosity

All the species involved in the simulation are stored in a relatively small STL vector called `speciesArray`, and every particle stores the index of the specie it adopts. The size of the `speciesArray` is only as large as the amount of species involved in the simulation, as specified by the input file.

3.2.3 Particle-Cell Linking

A linking procedure is created in order to unify, or "link", the computational domain to the particles in the domain such that the particles know which cell they are in and the cells know which particles they house. Thus, the child cell and particle data structures store this data.

The process begins by cycling through the particles that are initialized in the domain. The position of each particle is sent as an argument to the function

findChildCellFromLocation() which efficiently traverses the octree to return the leaf cell in which the particle resides.

In order for the leaf cells to access all the particles inside of it, the child cell must start by accessing the firstParticle integer then follow the linked list until the value of -1 is reached, which denotes the end of the linked list. This method is illustrated in Figure 3.11. This method is preferable over having each child store every particle in its space since that would involve complex memory adjustments of each child cell data structure, which could ultimately vary the size of the childCells vector at every time step.

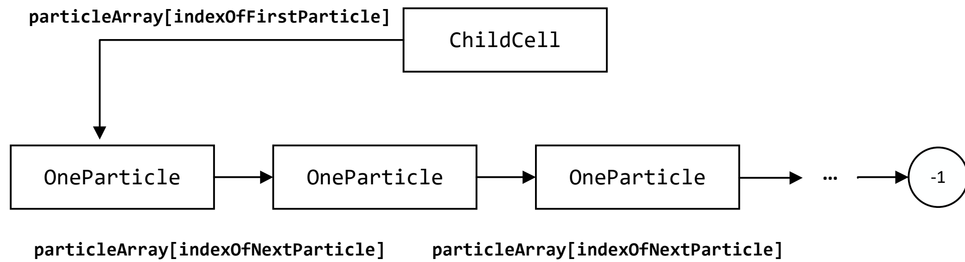


Figure 3.11 – Diagram of Particle-Cell Linking

3.3 SINATRA Classes

With regards to the second concept, the code is divided up into several classes that store data for and functionally control relevant pieces of the code. Table 3.9 below summarizes the classes used in SINATRA and includes a brief description of their purpose.

Table 3.9 – Summary of classes in SINATRA.

Class	Description
`Simulation`	Declares all other classes, controls the work flow of the simulation
`Initialization`	Reads input file and initializes key variables
`Mesh`	Stores and controls parent and child cell structs in octree structure
`Particle`	Stores and controls particle structs
`Kinematics`	Controls movement of particles, wall interactions, and time loop
`Collision`	Controls particle collisions
`Output`	Samples flow field for properties, writes all output files

There is one "main" function written in a file that can be named anything but is currently in the file called Development7.cpp. Here the name of the mesh file and input file are specified. Otherwise control is sent over to the Simulation class, which serves as the main driver for the DSMC simulation. The simulation is first setup by calling the Initialization class, then the Mesh and Particle classes use details from the Initialization class to produce the computational domain and the particles that will participate in the simulation. Once the simulation is set up, the Kinematics and Collision classes control the actual physics of the code. Finally, the Output class samples properties and creates all relevant output files.

Although this section provided an outline of the structure of the code itself, the actual implementation details of the code are discussed in Chapter 4. Furthermore, a more in depth discussion of the code itself can be found in the Developer's Notes section of the User's Manual.

CHAPTER 4

SINATRA Code Implementation

4.1 SINATRA Simulation Overview

Although SINATRA can perform basic DSMC simulations, there a several key simplifying assumptions underlying the capabilities of the code:

- Simple Cartesian mesh with limited refinement to run simulations on
- Electronic energy is neglected
- Binary elastic collisions only (no energy exchange during collisions)
- Discrete sub-cell method for collisions
- No boundary property sampling (flow property sampling only)
- No body forces

In order to run a simulation using SINATRA, only two text files are required: an input file defining the details of the simulation and a mesh file defining the grid that the simulation is performed on. The details on how the input file needs to be formatted is given in the User's Manual and the grid is formatted in Tecplot 360's 3D finite element ASCII grid format. After reading in these text files, the classes that compose SINATRA take over and run the simulation. These steps, and the classes involved in each step, can be seen in Appendix B. The equations implemented in the code to perform each step are outlined in the rest of this chapter.

4.2 Flow Initialization

In order to initialize the flow field, simulation particles need to have an initial position, velocity, and energy. At its current state, SINATRA only accounts for translational energy (which is a result of its velocity), rotational energy, and vibrational energy. Electronic energy is ignored, but would need to be taken into account when simulating chemical reactions.

To initialize the flow, the proper amount of particles are generated and appended to the `particleArray` vector. The initial flow number density and number fractions of each species specified in the input file is used to generate the proper number of particles. The total amount of particles to initialize is found by:

$$N_{total} = n_{mixture} V_{domain} \tag{4.1}$$

where N_{total} is the total number of particles, $n_{mixture}$ is the number density of the initial flow as specified in the input file, and V_{domain} is the volume of the entire simulation domain. The amount of each species to initialize is found by:

$$N_p = n_p N_{total} \tag{4.2}$$

where N_p is the total number of particles to initialize of species p and n_p is the number fraction of species p as specified in the input file.

4.2.1 Initial Position

Particles start off by being randomly distributed over the domain. All particles in the particle list are looped through and the following equation is applied:

$$
\begin{aligned}
x_{init} &= x_{low} + \texttt{rand01}(x_{high} - x_{low}) \\
y_{init} &= y_{low} + \texttt{rand01}(y_{high} - y_{low}) \\
z_{init} &= z_{low} + \texttt{rand01}(z_{high} - z_{low})
\end{aligned}
\tag{4.3}
$$

where the subscript *init* denotes the initial position, the subscripts *low* and *high* denote the value of the lowest and highest coordinate in each respective coordinate direction, and `rand01` represents a random fraction between 0 and 1. Many stochastic operations rely on a random fraction, so the function `rand01` was created as a shorthand way to return this value.

It is worth noting that particles may be more accurately represented by being initially uniformly spaced rather than randomly scattered throughout the domain. Further discussion of this will be given in Chapter 7.

4.2.2 Initial Velocity

The initial velocity components of the simulated molecules in an equilibrium gas are generated from the cumulative distribution function:

$$F_{\beta u'} = \{1 + \mathrm{erf}(\beta u')\}/2 \tag{4.4}$$

where $F_{\beta u'}$ is a random number from 0 to 1 and β is computed as $(m/2kT)^{1/2}$. Although the error function has no closed form inverse, it can be numerically approximated to sufficient accuracy as shown in reference [52]. This formed the foundation for a function called `erfinv()`. Thus, rather than using an accept-reject procedure, Equation 4.4 was inverted to give:

$$u' = \texttt{erfinv}(2F_{\beta u'} - 1)/\beta \tag{4.5}$$

Recall that u' is the thermal velocity component of a molecule and is related to the stream velocity u_0 by $u' = u - u_0$.

4.2.3 Initial Rotational Energy

In SINATRA, the rotational energy of molecules is initialized based on how many atoms compose the molecule. Monatomic molecules do not have rotational energy, so all monatomic molecules have zero rotational energy. Diatomic molecules, with two rotational degrees of freedom, are initialized using:

$$\varepsilon_{rot} = -\ln(R_F)kT \tag{4.6}$$

where k is the Boltzmann constant and R_F is a random fraction between 0 and 1. However, for polyatomic molecules, an accept-reject procedure must be followed. In order to effectively perform an accept-reject procedure, the ratio of the probability to the maximum probability for the property in question must be known. For the rotational energy of a polyatomic molecule, the ratio is

$$\frac{P}{P_{max}} = \left\{\frac{\varepsilon_{rot}}{(\zeta_{rot}/2 - 1)kT}\right\}^{\zeta_{rot}/2-1} \exp\left(\frac{\zeta_{rot}}{2} - 1 - \frac{\varepsilon_{rot}}{kT}\right) \tag{4.7}$$

To start the accept-reject method, a random ε_{rot} must be selected. In a perfect world, this random ε_{rot} should be any value from 0 to ∞. However, in order to obtain a realistic value and to assist in computation time, cut-off limits for ε_{rot} must be set. In SINATRA, the lower bound was set at 0.0 and the upper bound was arbitrarily set at $10kT(\zeta_{rot}/2)$, which is ten times greater than the most probable value.

After the random ε_{rot} is selected, it is inserted into Equation 4.7 to obtain a value of P/P_{max}. This value is then compared to another random fraction,R_F, between 0 and 1, and if P/P_{max} is greater than R_F, then the initial ε_{rot} is "accepted." Otherwise, a new ε_{rot} is sampled and the procedure starts all over.

4.2.4 Initial Vibrational Energy

As mentioned in Section 2.3 the values for vibrational energy states are spaced much further apart than rotational energy states. While the values for rotational energy are generated on a continuous fashion, vibrational energy is not.

The equation for the vibrational energy of a specific mode can be initially found with the same equation used for rotational energy of a diatomic molecule (4.6). The only added procedure is to truncate this value to $ik\Theta_{vib}$ of the highest available level (where i is an integer value). The solution proceeds as follows:

Since the equations start by

$$ik\Theta_{vib} = -\ln(R_F)kT \tag{4.8}$$

The exact vibrational level can be solved by

$$i = \texttt{floor}(-\ln(R_F)T/\Theta_{vib}) \tag{4.9}$$

In contrast to Bird's code [44] which stores the vibrational energy level, i, of each particle, SINATRA stores the actual vibrational energy value, ε_{vib}, resulting from the vibrational energy level. Regardless, both can readily be found from one another by using Equations 4.8 and 4.9.

4.3 Particle Advection

Simple procedures are used to advect particles at each time step. Since gravity or other body forces are not taken into account in this code, particles do not accelerate and are assumed to move in a direct path across the domain unless undergoing a collision. Thus, the equations used to update a particles positions are:

$$x_{new} = x_{prev} + u\Delta t \tag{4.10}$$

$$y_{new} = y_{prev} + v\Delta t \tag{4.11}$$

$$z_{new} = z_{prev} + w\Delta t \tag{4.12}$$

where the subscript *prev* denotes the original coordinate position and the subscript *new* denotes the updated coordinate position after time step, Δt. Here, u, v, and w are the velocities in the x, y, and z directions, respectively.

4.4 Boundary Conditions

In order to properly capture realistic flow conditions, several boundary conditions were developed.

4.4.1 Inlet

One of the most important capabilities of a computational fluid dynamics (CFD) code is to incorporate an inlet, which allows molecules (and ultimately gas) to flow into the computational domain.

$$\dot{N} = n\Big[\exp\left(-{s_n}^2\right) + \pi^{\frac{1}{2}} s_n \big\{1 + \mathrm{erf}(s_n)\big\}\Big] \Big/ \big(2\pi^{\frac{1}{2}}\beta\big) \tag{4.13}$$

where

$$\beta = \big\{m/\big(2kT\big)\big\}^{\frac{1}{2}} \tag{4.14}$$

and s_n is the "speed ratio" given by:

$$s_n = u_0\beta \tag{4.15}$$

Note that for $u_0 = 0$, as is the case for an inlet with a stationary gas, Equation 4.13 reduces to

$$\dot{N} = n / \left(2\pi^{\frac{1}{2}}\beta\right) \tag{4.16}$$

Once the number of particles to insert is calculated, the particles are assigned a velocity using the ratio of the probability to maximum probability of a thermal velocity given by:

$$\frac{P}{P_{max}} = \frac{2(\beta u' + s_n)}{s_n + (s_n^2 + 2)} exp\left[\frac{1}{2} + \frac{s_n}{2}\left\{s_n - \left(s_n^2 + 2\right)^{\frac{1}{2}}\right\} - \beta^2 u'^2\right] \tag{4.17}$$

The accept-reject method is used so find velocity component normal to the face that the particle is entering through. It is worth noting that in order for the assigned velocity to make physical sense, the prescribed velocity for the particle must point toward the flow in the computational domain. This is done by placing limits on the value for $\beta u'$ and ensuring that $\beta u' + \beta u_0 \geq 0$ (or $\beta u' + s_n \geq 0$).

4.4.2 Outlet

Recall that each of the components of a gas molecule's velocity are statistically centered around a specific velocity, often the free stream velocity, u_0. This means that although the flow may be specified has having a stream velocity of 100 m/s in the positive x-direction, there still may be particles that have a velocity component in the negative x-direction. Thus, inlets can act as outlets and outlets can act as inlets. This means both inlets and outlets are considered "reservoir" boundary conditions and the same equations are applied for both (see Section 4.4.1).

4.4.3 Specular Wall

A specular wall acts as a plane of symmetry in that the velocity component normal to the surface is reversed after the collision.

An impact of a particle and a specular wall is considered and elastic collision since momentum is conserved and energy levels do not change.

A simple procedure is used to handle specular wall boundary conditions:

1. Detect particle has crossed a specular wall
2. Compute the fraction of the overall time step used to intersect that wall
3. The rest of this time step is saved in the particle data structure as `dtremain`
4. Particle is then placed at the point of intersection of the boundary
5. Velocity component normal to the boundary is reversed
6. Particle is advected for the rest of `dtremain`

4.4.4 Diffuse Wall

The diffuse wall boundary condition attempts to better simulate actual particle-surface interactions. The properties of particles after colliding with diffuse walls are as if they were effusing from a fictitious gas at the surface temperature on the opposite side of the surface [44]. As such, the particles' properties follow the same equations as Inlet particles given in Section 4.4.1. The only difference is that now these equations are modified to account for the fact that the diffuse wall may have a specified temperature and velocity. Thus, this boundary interaction is inelastic in almost all situations as the velocity is changed. The procedure used to handle particle interactions with diffuse wall is similar to the procedure used in specular walls, except in how the new velocity is computed. Otherwise, the procedure used for specular walls is followed.

4.4.5 Periodic Wall

Periodic walls (also known as cyclic walls) simply move exiting particles on one end of the domain to the other side.

4.5 Particle-Cell Re-Linking

After the particles have moved, been injected into the domain, left the domain, and/or interacted with boundary walls, they are re-linked to the cells so that the particles know which cell they are in and the cells know which particles are inside them.

Immediately after particle advection, the new particle location is checked. If the particle left its original cell, it is added to a list of particles that must be re-linked. This re-linking occurs after any potential boundary interactions take place. It is only those particles that move to a new cell that get re-linked, so the computation time required is not nearly as long as the initial particle-cell linking at the beginning of the simulation.

4.6 Particle Collisions

The Discrete Sub-Cell/No Time Counter (NTC) method is currently the only developed method in SINATRA and is one of the oldest DSMC collision methods used. The Discrete Sub-Cell method works by defining collision cells at the beginning of the simulation and using the collision cell's eight children as "sub-cells" where collisions are narrowed.

During the collision phase, every collision cell is visited and the amount of potential collisions occurring in a given time step is calculated using Equation 4.19 to calculate the collision cross section, Equation 4.22 to calculate the relative velocity, and Equation 4.18 to calculate the collision rate,

$$\frac{1}{2}N(N-1)F_N(\sigma c_r)_{max}\frac{\Delta t_{cell}}{V_{cell}} \tag{4.18}$$

where Δt_{cell} is the time step, V_{cell} is the volume, and N is the number of particles of the collision cell in question. In general, σ is the collision cross section and c_r is the relative velocity between two particles, however in this case, their product is relevant and the maximum value of this product is stored. In SINATRA, every collision cell has the same time step, so in the current version of SINATRA, Δt_{cell} can be replaced by Δt. However, other DSMC methods employ variable time stepping schemes which are described in Section 7.8.

The collision cross section is computed by:

$$\sigma = \pi(\frac{1}{2}(d_1 + d_2))^2 \tag{4.19}$$

where the diameter, d, depends on the sphere model specified and is computed using the following pseudo-algorithm:

```
if Hard Sphere
```

$$d_{HS} = \sqrt{\frac{5}{16\mu_{ref}}\sqrt{\frac{mkT_{ref}}{\pi}}} \tag{4.20}$$

```
else if Variable Hard Sphere
```

$$d_{VHS} = d_{ref}\sqrt{\frac{\left(\frac{2kT_{ref}}{m_r c_r^2}\right)^{\omega-1/2}}{\Gamma(\frac{5}{2}-\omega)}} \tag{4.21}$$

where d_{ref}, μ_{ref}, T_{ref} and ω are properties specific to the species of each particle, and the relative velocity, c_r is computed by:

$$c_r = \sqrt{(u_2-u_1)^2 + (v_2-v_1)^2 + (w_2-w_1)^2} \tag{4.22}$$

Note that the subscripts 1 and 2 refer to each of the two particles arbitrarily labeled as 1 and 2. Based on the amount of possible collisions, a random particle within the collision cell is selected. Then, based on the sub-cell that the randomly selected particle is in, another particle is selected as the original particle's potential partner. If there are no more particles within a sub-cell, other sub-cells are querried within the collision cell. Once collision pairs are identified, collision pairs are approved and ultimately collided based on the accept-reject method given by Equation 4.23.

$$\frac{P}{P_{max}} = \frac{\sigma c_r}{(\sigma c_r)_{max}} \tag{4.23}$$

The $(\sigma c_r)_{max}$ term is inherent in the collision cell and is stored in the `sigmacrmax` variable stored in the child cell struct as shown in Section 3.2.1. This value is potentially different from one collision cell to another. Also, this value is updated when a higher σc_r value is calculated.

Once the collision pairs are identified and the collision is approved during the accept-reject method, the velocities are updated by first finding the velocity of the center of mass between the two particles undergoing the collision:

$$c_{m,x} = \frac{m_1}{m_1 + m_2} u_1 + \frac{m_2}{m_1 + m_2} u_2 \tag{4.24}$$

$$c_{m,y} = \frac{m_1}{m_1 + m_2} v_1 + \frac{m_2}{m_1 + m_2} v_2 \tag{4.25}$$

$$c_{m,z} = \frac{m_1}{m_1 + m_2} w_1 + \frac{m_2}{m_1 + m_2} w_2 \tag{4.26}$$

Next, the post-collision deflection angles are computed by:

$$\cos(\chi) = 2\texttt{rand01()} - 1 \tag{4.27}$$

$$\sin(\chi) = \sqrt{1 - (\cos(\chi))^2} \tag{4.28}$$

$$\phi = 2\pi \texttt{rand01()} \tag{4.29}$$

The post-collision, denoted by the superscript star, relative velocity components are computed by:

$$u_r^* = c_r cos(\chi) \tag{4.30}$$

$$v_r^* = c_r sin(\chi) cos(\phi) \tag{4.31}$$

$$w_r^* = c_r sin(\chi) sin(\phi) \tag{4.32}$$

Finally, the post-collision velocities of the first particle is computed by:

$$u_1^* = c_{m,x} + \frac{m_2}{m_1 + m_2} u_r^* \tag{4.33}$$

$$v_1^* = c_{m,y} + \frac{m_2}{m_1 + m_2} v_r^* \tag{4.34}$$

$$w_1^* = c_{m,z} + \frac{m_2}{m_1 + m_2} w_r^* \tag{4.35}$$

and for the second particle:

$$u_2^* = c_{m,x} - \frac{m_1}{m_1 + m_2} u_r^* \tag{4.36}$$

$$v_2^* = c_{m,y} - \frac{m_1}{m_1 + m_2} v_r^* \tag{4.37}$$

$$w_2^* = c_{m,z} - \frac{m_1}{m_1 + m_2} w_r^* \tag{4.38}$$

After particles have collided, those particular particles are no longer eligible for collisions in a given time step and are removed from the list of possible collision particles. This remains consistent with the concept of "binary collisions".

It makes no sense to have more collision pairs than available particles in the collision cell. Thus, this is the driving factor in determining the simulation parameters,

most notably the time step and FNUM. An error message will appear if there are not enough particles in the collision cell to form a realistic number of collision pairs.

Collisions only affect the particles' velocity. This effect is noticed when advection repeats during the next time step.

4.7 Flow Property Extraction

As noted earlier in the introduction, the magic of molecular-scale methods is that macroscopic properties (those that humans experience everyday) can be extracted from microscopic properties and behavior. In order to perform this same extraction from simulated molecules in the code, the basic equations from Chapter 2 needed to be utilized and adapted for the existing code structure. This is accomplished by using "sampling cells" which are existing cell structures for which flow properties are extracted. These can, but do not necessarily, differ from "leaf cells" that contain the actual particle data and/or "collision cells" for which collisions are computed. For the sake of simplicity and for lack of evidence otherwise, the sampling cells used in SINATRA are the same as the collision cells.

All the equations in this section are taken directly from [44] unless otherwise specified. These equations use the term "weighting factor" which are often used for simulations with small, trace species or axisymmetric DSMC simulations (see discussion in Section 10.4 of [11]). These weighting factors have not been incorporated in SINATRA.

4.7.1 Density

The density of a cell is the easiest flow property to visualize and the most straighforward to calculate. The gas density of each cell is:

$$\rho = n \sum_{p=1}^{q} (m_p N_p) \Big/ \sum_{p=1}^{q} N_p \tag{4.39}$$

4.7.2 Velocity

The velocity is computed by:

$$\begin{aligned} u_0 &= \sum_{p=1}^{q} (m_p \sum^{N_p} u_p) \Big/ \sum_{p=1}^{q} (m_p N_p) \\ v_0 &= \sum_{p=1}^{q} (m_p \sum^{N_p} v_p) \Big/ \sum_{p=1}^{q} (m_p N_p) \\ w_0 &= \sum_{p=1}^{q} (m_p \sum^{N_p} w_p) \Big/ \sum_{p=1}^{q} (m_p N_p) \end{aligned} \tag{4.40}$$

4.7.3 Temperature

Temperature is slightly more complicated to calculate. Every species contributes is own temperature to the overall gas mixture temperature. Furthermore, within every species temperature there are components contributed from translational energy, rotational energy, and vibrational energy.

SINATRA first calculates each species temperature from the three different energy components, then calculates the overall mixture temperature from every species.

To calculate the translational temperature of species p:

$$T_{tr,p} = m_p \Big\{ \sum^{N_p} (u_p^2) + \sum^{N_p} (v_p^2) + \sum^{N_p} (w_p^2) - u_0^2 - v_0^2 - w_0^2 \Big\} \Big/ (3k) \tag{4.41}$$

where u_0, v_0, and w_0 are the free stream velocities of the sampling cell calculated in Equations 4.40. Notice that these equations imply that there is a temperature component associated with how the particles' velocity deviates from the flow velocity.

To calculate the rotational temperature of species p:

$$T_{rot,p} = (2/k)\Big(\sum^{N_p} \varepsilon_{rot,p}/\zeta_{rot,p}\Big) \tag{4.42}$$

where ε_{rot} is the rotational energy stored in the particle structure and $\zeta_{rot,p}$ is the number of rotational degrees of freedom of the species stored in the species structure.

Recall from Section 2.3 that molecules can have multiple modes of vibrational energy. Thus the vibrational temperature of every species p must include the vibrational temperature contribution from every single mode.

To calculate the vibrational temperature of mode l of species p:

$$T_{vib,l,p} = \Theta_{vib,l,p}/\ln(1 + N_p/\sum(i^l_{vib,p})) \tag{4.43}$$

where $\Theta_{vib,l,p}$ is the characteristic vibrational temperature of mode l of species p. This value is specified for each species by the user in the Input File and is stored in the `Species` structure in code.

The number of effective vibrational degrees of freedom of each mode l of species p is:

$$\zeta_{vib,l,p} = 2(\sum(i^l_{vib,p})/N_p)\ln(1 + N_p/\sum(i^l_{vib,p})) \tag{4.44}$$

and the total number of vibrational degrees of freedom of species p is calculated by summing $\zeta_{vib,l,p}$ over all m modes:

$$\zeta_{vib,p} = \sum_{l=1}^{m} \zeta_{vib,l,p} \tag{4.45}$$

To calculate the vibrational temperature of species p:

$$T_{vib,p} = \sum_{l=1}^{m} (\zeta_{vib,l,p} T_{vib,l,p}) \big/ \zeta_{vib,p} \tag{4.46}$$

SINATRA is currently only compatible with simple molecules with one vibrational energy mode, thus the equation for vibrational temperature in SINATRA is simply Equation 4.43 without the l subscript.

Once the individual temperature components from each type of energy are calculated for one species, the effective number of degrees of freedom and the temperature of species p must be computed.

The effective number of degrees of freedom of species p is:

$$\zeta_p = 3 + \zeta_{rot,p} + \zeta_{vib,p} \tag{4.47}$$

and the temperature of species p is calculated by:

$$T_p = \big(3T_{tr,p} + \zeta_{rot,p} T_{rot,p} + \zeta_{vib,p} T_{vib,p}\big) \big/ \zeta_p \tag{4.48}$$

where the 3 in Equation 4.47 and the coefficient 3 before $T_{tr,p}$ in Equation 4.48 represents the three coordinate directions and is constant.

Finally, the overall gas temperature of a cell can be calculated by:

$$T = \sum_{p=1}^{q} (\zeta_p T_p N_p) \Big/ \sum_{p=1}^{q} (\zeta_p N_p) \tag{4.49}$$

4.7.4 Pressure

The computation for the pressure at each cell can be found by summing the pressure contributions from each species p.

$$p_p = (N_p/V_{cell}) F_{NUM} k T_{tr} \tag{4.50}$$

and the overall gas pressure at each cell can be computed by:

$$p = \sum_{p=1}^{q} (p_p) \tag{4.51}$$

4.8 Post Processing

4.8.1 Tecplot

All property illustrations were extracted and exported to Tecplot 360™. A comprehensive outline of how data needs to be written to be compatible with the Tecplot reader can be found in [53]. The start time and frequency of when these data files are generated are specified by the user in the input file.

4.8.2 MATLAB

MATLAB was primarily used for visualization purposes. One of the key abilities, not pertanent to DSMC in general, but visually appealing and helpful for debugging is the script that animates all of the particles (shown in Appendix D). Note that these animations are generated by displaying a three-dimensional scatter plot of every particle's position throughout the domain at every time step. The output files used to generate this contain every particle's position, velocity, and species. There is one of these generated files for every time step specified, so storage of these files can significantly add up depending on the number of particles and length of simulation. This is not recommended for large simulations. The start time and frequency of when these data files are generated are specified by the user in the input file.

CHAPTER 5

Code Validation and Test Cases

5.1 Random Number Generator

Many of the equations native to the DSMC method, and stochastic processes as a whole, rely heavily on a random fraction between 0 and 1. In SINATRA, the function `rand01()` was created as a shorthand way to produce this number.

In order to test the validity of `rand01()` in SINATRA and the ability to produce sufficiently random numbers a qualitative test case was run that mimics a test case from [54]. The first test case was to produce 1 million pseudo random numbers between 0 and 1. The results are shown below in Figure 5.1.

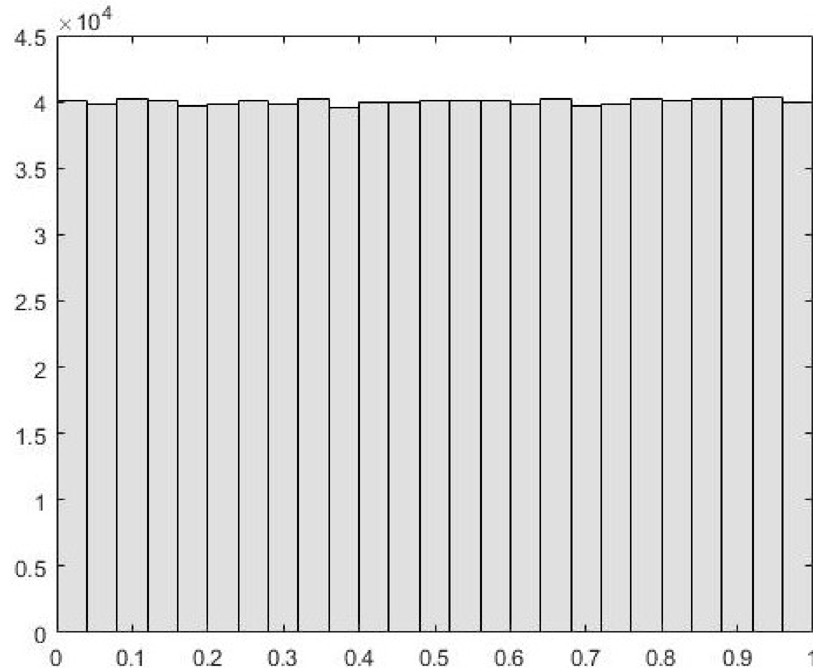


Figure 5.1 – Histogram of a sample of 1 million pseudo-random points.

Figure 5.1 implies good qualitative results as no small range of numbers is generated significantly more than others.

Another consideration when testing the random number generator is also ensuring that it has been seeded correctly. Since the random number generator is not truly random, but pseudo-random, the same set of random numbers will appear in the same order if they have the same seed. To ensure the random numbers did not repeat, the random number generator was repeatedly called at the same location after the seed, and results were compared to ensure the numbers did not repeat.

Successful tests of the random number generator provides the user confidence that SINATRA's results will actually model stochastic processes.

5.2 Initial Flow Field Properties

Before any of the physical interactions involved in DSMC begin, it is important to ensure that the flow field is initialized correctly. The properties analyzed to ensure the flow is initialized correctly are the positions, velocities, and energy levels of the particles.

5.2.1 Initial Position

Since the positions of each particle are initialized randomly throughout the domain, and therefore depend on the random number generator, the behavior of this initialization is again investigated qualitatively. This test is motivated from a study in [54]. Here, 5000 particles were initialized throughout a 1m x 1m x 1m domain. This was performed several times, and although the results were different every time, they were qualitatively similar in terms of scatter. Thus only one of the results is shown below in Figure 5.2.

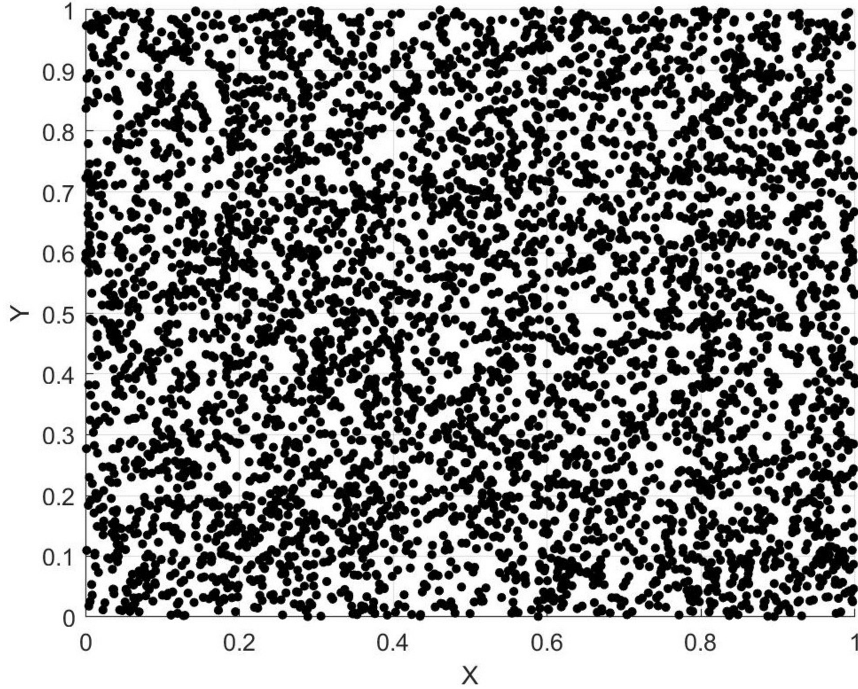


Figure 5.2 – Projection of the initial positions of particles when viewed through a 2D plane.

Despite the fact that SINATRA is a 3D code, the results were taken and viewed through a 2D plane perpendicular to the z-axis. Thus, only the x and y positions are viewed.

Although there is comfort knowing that the particles are distributed at locations throughout the entire domain, there is noticeable grouping of particles at certain x, y locations and gaps at other locations. A perfectly uniform distribution is not expected as positions are assigned randomly. However, this may indeed cause *excessive* scatter as warned by Bird [44] Section 4.1.

5.2.2 Initial Velocity

In order to test if the velocities of the initialized particles follow the theoretical values given by the probability distribution functions, data was pulled from SINATRA,

normalized, and overlayed on the theoretical curves. The average value of SINATRA's initial velocity was compared to the theoretical value for varying quantities of initialized particles.

A stationary gas flow field was initialized with the properties shown below in Table 5.1 and Figure 5.3 below shows the initial distribution of all velocity components of a stationary gas.

Table 5.1 – Summary of flow properties for velocity distribution analysis.

Property	Value
Species	N_2
Mol. Mass (kg)	4.65 x 10^{-26}
Mol. Diam. (m)	4.17 x 10^{-10}
Temperature (K)	300
u_0 (m/s)	0 m/s
v_0 (m/s)	0 m/s
w_0 (m/s)	0 m/s

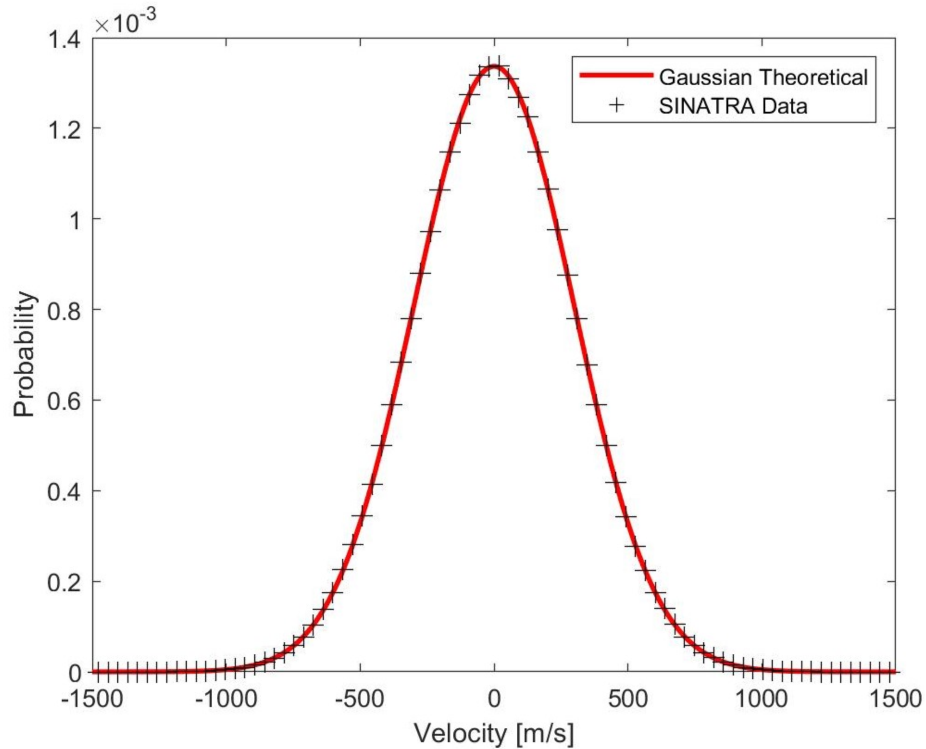


Figure 5.3 – SINATRA data of all three initial velocity components overlayed on probability distribution function.

Figure 5.3 shows good agreement between SINATRA's initialized particle velocities and the theoretical. Each velocity component was supposed to be centered around 0 m/s and the average of all the velocity components is -7.9 x 10^{-4} which can be safely approximated as 0.

Taking the magnitude of the speed of every particle (from its three velocity components) and plotting their probabilities gives what is known as the Maxwellian (or Boltzmann) distribution of particle speed, as shown below in Figure 5.4.

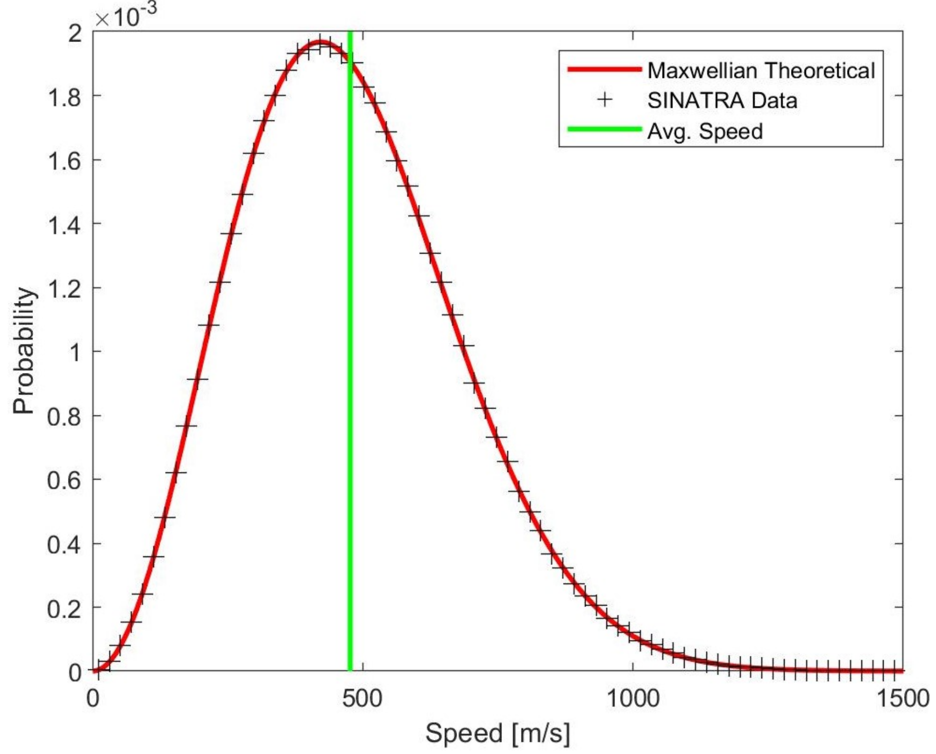


Figure 5.4 – SINATRA initial speed data overlayed on probability distribution function yielding a Maxwellian distribution.

The most probable speed corresponds to the peak value of the Maxwellian curve and is theoretically calculated by:

$$c'_m = \frac{1}{\beta} = \sqrt{\frac{2kT}{m}} \tag{5.1}$$

which evaluates to 422.1 m/s in this particular flow field.

An alternative way to quantify the accuracy of the code is by comparing the values of the average molecular speed (which is a scalar multiple of the most probable speed) between theoretical and SINATRA data values. Theoretically, this is calculated by:

$$\bar{c}' = \frac{2}{\sqrt{\pi}\beta} = \frac{2c'_m}{\sqrt{\pi}} \tag{5.2}$$

For SINATRA, this is represented by the vertical green line in Figure 5.4 and is a better predictor of accuracy since the mean value of SINATRA's data is not skewed by the width of bin sizes when generating the probability distribution, like comparison of the most probable speed would be.

The SINATRA data points in Figures 5.3 and 5.4 were generated using 1 million particles and the percent error from theoretical is quite good (0.04%). Table 5.2 shows how this percent error changes when the number of initialized particles change.

Table 5.2 – Effect of the number of particles on the percent error of the average molecular speed of the system.

Number of Particles	Percent Error (%)
100	8.87
1,000	0.66
10,000	0.18
100,000	0.08
1,000,000	0.04

The table reveals that there is significant scatter still present when only 100 particles are initialized and sampled, but as expected, the results dramatically improved for a greater number of particles in the domain.

5.2.3 Initial Energy

A comparison similar to that used to analyze the initial velocity is also used to quantify the accuracy of the initial energy levels. The results are used to analyze the effectiveness of the procedures used to initialize energy values.

There is a key difference between the way rotational energy is initialized for diatomic and polyatomic molecules (recall that monotomic molecules have no rotational energy). While a simple equation utilizing a random fraction is used for diatomic

molecules, an accept-reject procedure is used for polyatomic molecules. Thus, results for both types of molecules need to be tested to ensure accuracy.

Figure 5.5 below shows the SINATRA rotational energy data overlayed on the probability distribution function for the diatomic molecule, Nitrogen (N_2). This flow utilizes the same properties shown in Table 5.1.

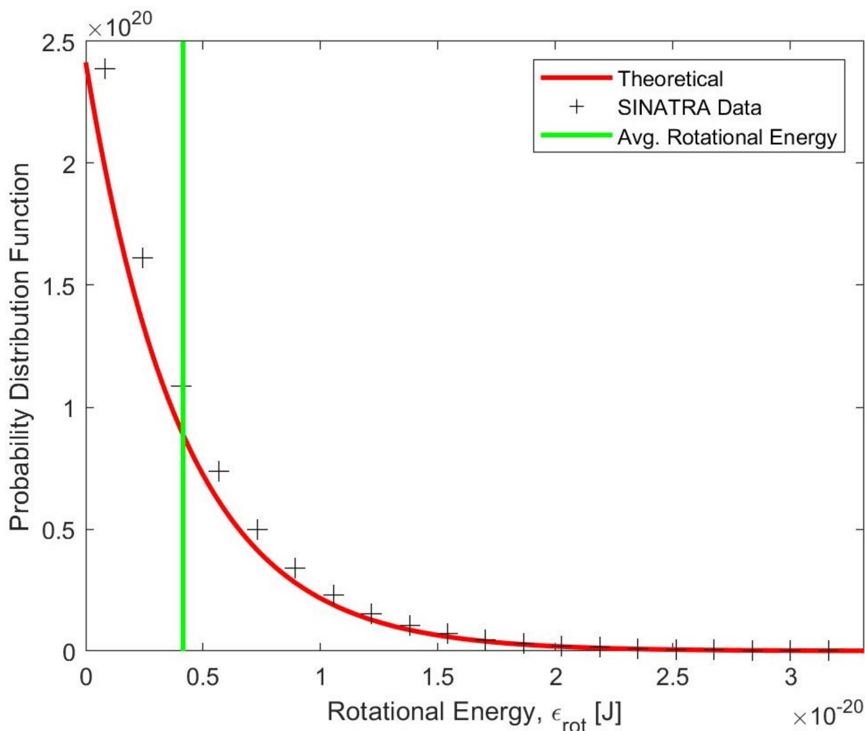


Figure 5.5 – SINATRA initial rotational energy overlayed on probability distribution function for N_2.

Figure 5.5 shows that the normalized distribution function taken from SINATRA data does not lie perfectly on the theoretical probability distribution function. However, Table 5.3 shows that the average rotational energy taken from SINATRA's data is actually in close agreement with the theoretical values, especially as the number of particles approaches one million. Here the theoretical value is taken as $\epsilon_{th} = kT$ since $\frac{1}{2}kT$ is contributed by each degree of freedom.

One possible reason for the imperfect overlay is the way that the distribution functions are created from SINATRA data. The rotational energy from every particle in the initial flow is stored, then those values are placed in bins to generate a histogram. The amount of particles in each bin is then divided by the area under the histogram curve (found through numerical integration) so that the probability distribution function integrates to unity over the entire domain. Since the energy values are so small in magnitude, the value of the initial numerical integration is subject to precision error which may yield seemingly imperfect results.

Table 5.3 – Effect of the number of particles on the average rotational energy for Nitrogen (diatomic molecule).

Number of Particles	Percent Error (%)
100	4.79
1,000	1.64
10,000	1.39
100,000	0.20
1,000,000	0.08

Figure 5.6 below shows the SINATRA rotational energy data overlayed on the probability distribution function for the polyatomic molecule, Carbon Dioxide (CO_2).

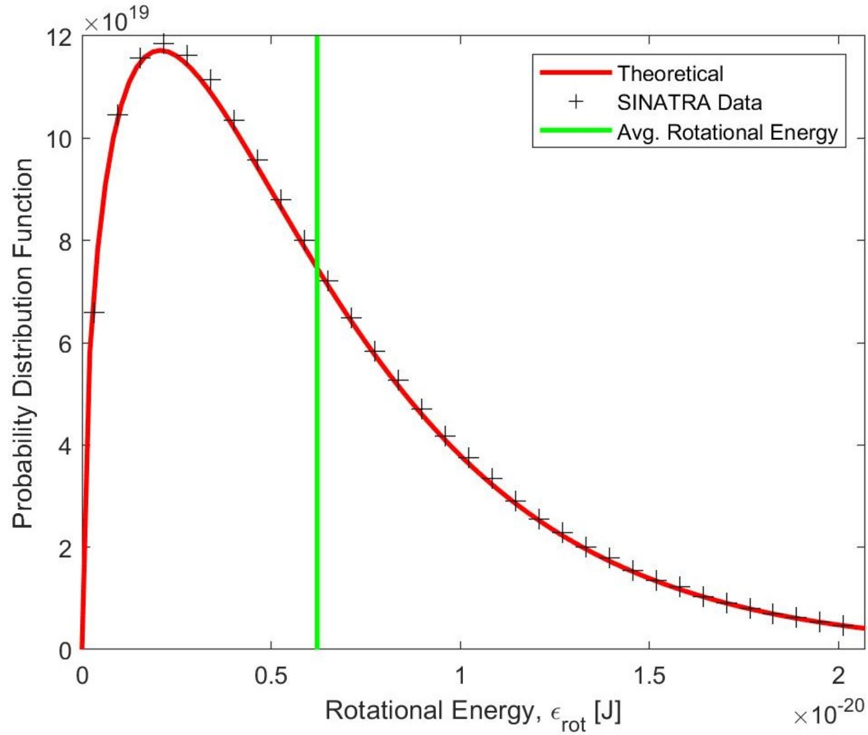


Figure 5.6 – SINATRA initial rotational energy overlayed on probability distribution function for CO_2.

The temperature and velocity components are the same as those shown in Table 5.1, but the species data is that which corresponds to Carbon Dioxide. This data is summarized in Table 5.4 below.

Table 5.4 – Summary of relevant species data for CO_2.

Species	Mol. Mass (kg)	Mol. Diameter (m)
CO_2	7.31 x 10^{-26}	5.62 x 10^{-10}

The percent error of the average rotational energy for polyatomic molecules is shown below in Table 5.5.

Table 5.5 – Effect of the number of particles on the average rotational energy for Carbon Dioxide (polyatomic molecule).

Number of Particles	Percent Error (%)
100	1.78
1,000	1.63
10,000	0.57
100,000	0.15
1,000,000	0.09

It appears that the accept-reject method used to initialize the rotational energies of polyatomic molecules works better than the random fraction equation used for diatomic molecules, both in visual adherence to the probability distribution function and percent error of the average value from theoretical for smaller quantities of particles.

Overall, SINATRA does a good job initializing the flow field at the beginning of DSMC simulations.

5.3 Property Sampling

One of the primary ways to post-process and validate simulation results are by extracting macroscopic, physical properties from the flow field. Since the DSMC method is highly stochastic, there is potential for these properties to be inaccurate if an insufficient number of particles are used to sample the flow field. Since these macroscopic properties are sampled from "sample cells", the number of particles in each sample cell necessary to obtain accurate results is explored.

In order to test this, a flow field is initialized at prescribed properties. These properties are summarized in Table 5.6. Note that the pressure is derived using the ideal gas law. The flow domain surrounded by diffuse walls with wall temperatures being equal to that of the flow itself. These are similar conditions to the initialized

flow being surrounded by stationary flow of the same properties. The input file used to run these results are shown in Appendix E.1. Figures 5.7, 5.8, 5.9, and 5.10 show how the results of the mesh convergence study, and how the sampled properties change base on the number of particles in each sample cell.

Table 5.6 – Prescribed properties used for mesh convergence study.

Property	Value
n (m^{-3})	1 x 10^{20}
Temperature (K)	273
Pressure (Pa)	0.3769
u (m/s)	0
v (m/s)	0
w (m/s)	0
c (m/s)	0

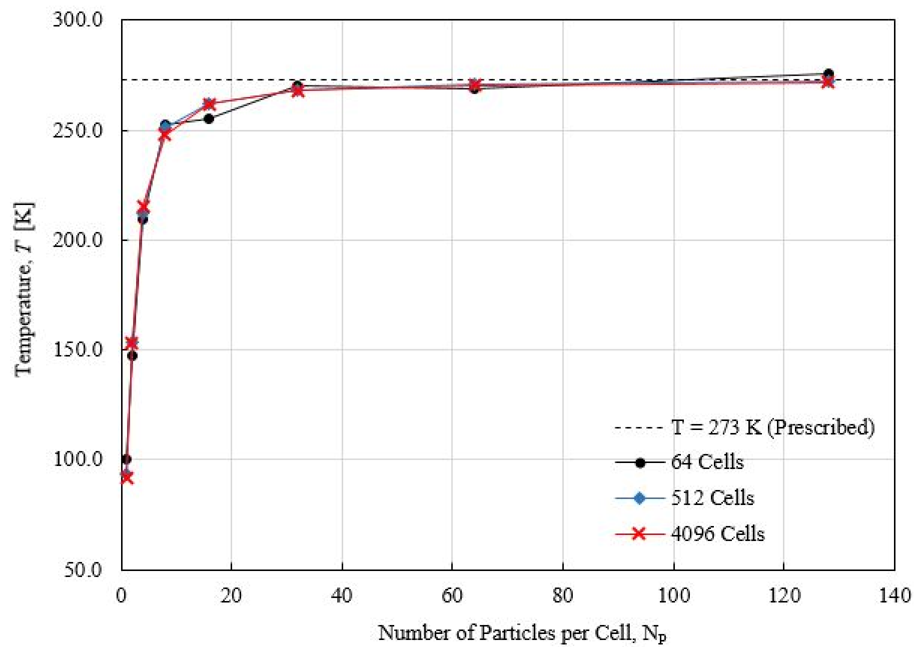


Figure 5.7 – Convergence study for temperature.

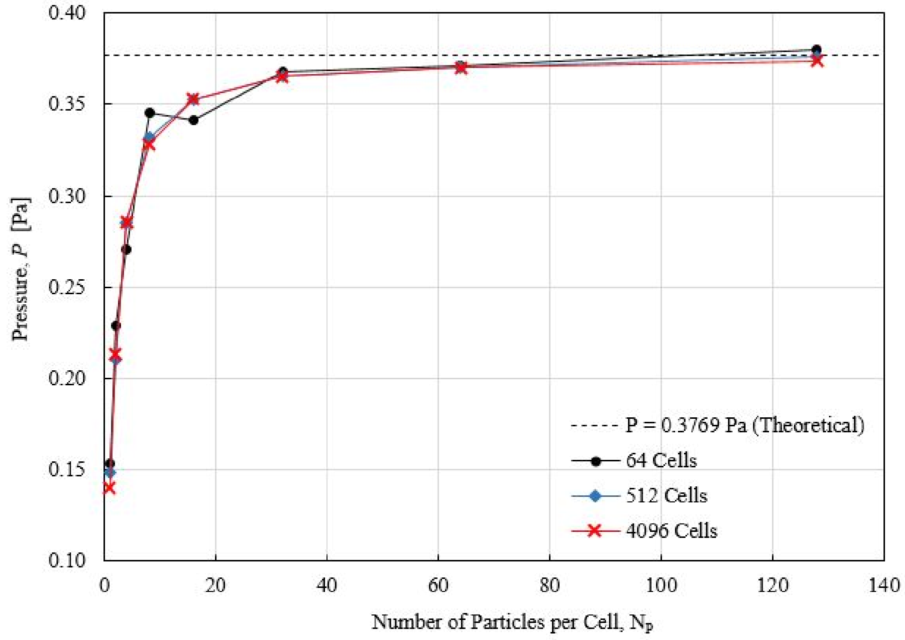


Figure 5.8 – Convergence study for pressure.

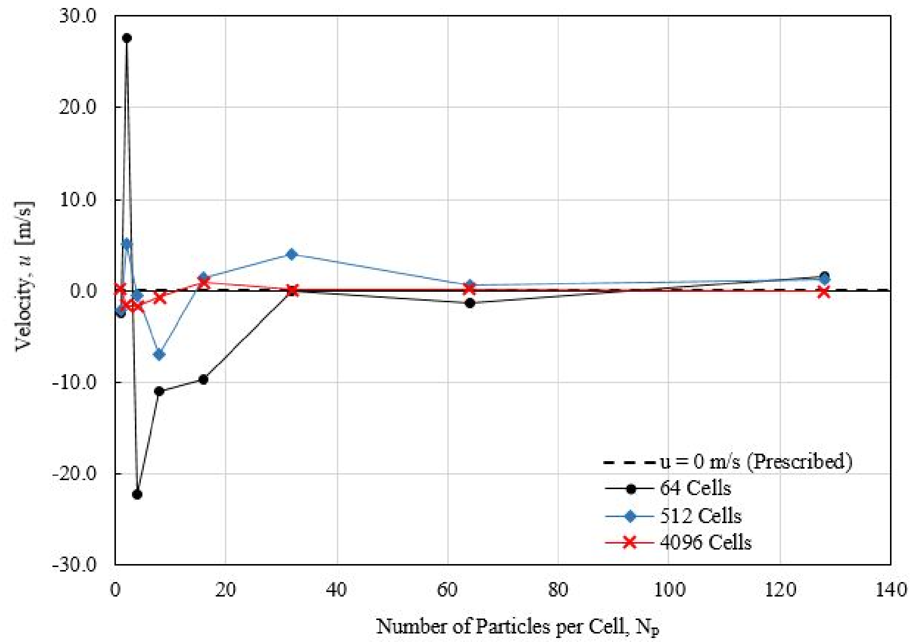


Figure 5.9 – Convergence study for velocity.

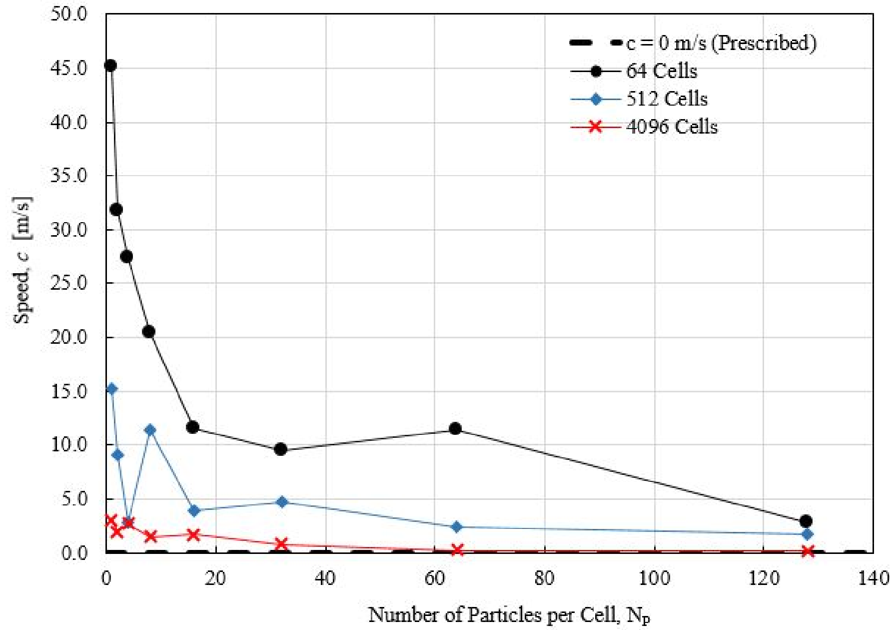


Figure 5.10 – Convergence study for speed.

The results shown in the above figures give very valuable information about the amount of particles required to be in a sample cell in order to extract the correct macroscopic properties from the simulation.

Figure 5.7 reveals that the proper temperature is found when there are approximately 35 particles in each sample cell. However, Figure 5.8 reveals that in order to obtain the proper pressure reading, there needs to be closer to 100 particles per sample cell to hit the exact value, though values come close at around 40 particles per cell. Figures 5.9 and 5.10 may give some insight as to why this occurs. Even after the results for velocity approach the prescribed value, there is a noticeable scatter about the correct result. Since almost all of the properties rely, to some extent, on the sampled velocity, it is important to ensure that the velocity sampled is as close to the actual value as possible. The speeds (which are a direct result of the velocities) for the coarser meshes appear to approach the correct value of 0 m/s beyond 120 particles per cell. This amount, of course, is not ideal. It does not appear the number of cells

to discretize the domain make much of a difference until the convergence study for velocity, and more obviously, speed, are seen. For these properties, the more refined mesh appears to cause the sampled properties to converge to the correct solution with much fewer particles. Thus, a highly refined mesh is recommended.

Since it appears that the velocity samples are much more sporadic and require more particles per cell to reach accurate values, it may beneficial to assign velocities differently. Some methods on how to do this are discussed in Section 7.1.

5.4 Collisionless Test Cases

If the mean free path, λ, and in turn the Knudsen number, is high enough, the flow is considered "collisionless" as the particles are unlikely to collide with one another. Naturally, this regime was developed before collisions are modeled and a few test cases were compared against Bird's code.

5.4.1 Statistical Scatter

This test case attempts to mimic the test case given in [44] Section 4.1, and Figure 5.11 attempts to mimic Figure 4.1 of that same book.

The goal of this test case is to show the effect of statistical scatter on the total number of particles in the domain for a steady flow. In turn, this also tests the inflow/outflow boundary conditions since those are the routines that cause this scatter.

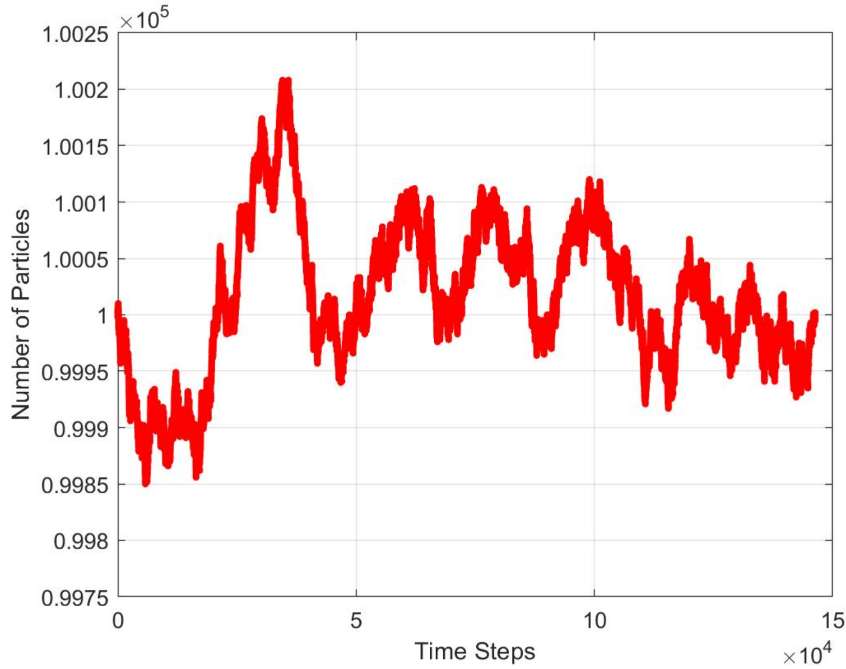


Figure 5.11 – The number of particles in the domain over time.

Although it appears that there is a lot of scatter, this is completely natural side effect of the DSMC method. In fact, the idea that this scattering oscillates about the prescribed amount of particles validates that the inflow and outflow boundary conditions are implemented correctly. Even though scattering is natural, the magnitude of scattering can be of concern. Table 5.7 below shows a comparison of the maximum scatter in SINATRA versus Bird's DSMC code used in [44]. These results show agreement between the two codes.

Table 5.7 – Comparison of percent variation between DSMC.F90 and SINATRA.

Percent Variation (%)	
DSMC.F90	SINATRA
±0.15	±0.21

5.4.2 Collisionless Heat Transfer

This particular test case mimics a one-dimensional steady flow test case given Section 7.3 of [11] which simulates the collisionless heat transfer between two infinite plane parallel plates at different temperatures and separated by a distance, h. A diagram of the simulation layout can be seen below in Figure 5.12 and the input file used for the simulation can be found in Appendix E.2 with only the upper plate temperature varying in each simulation.

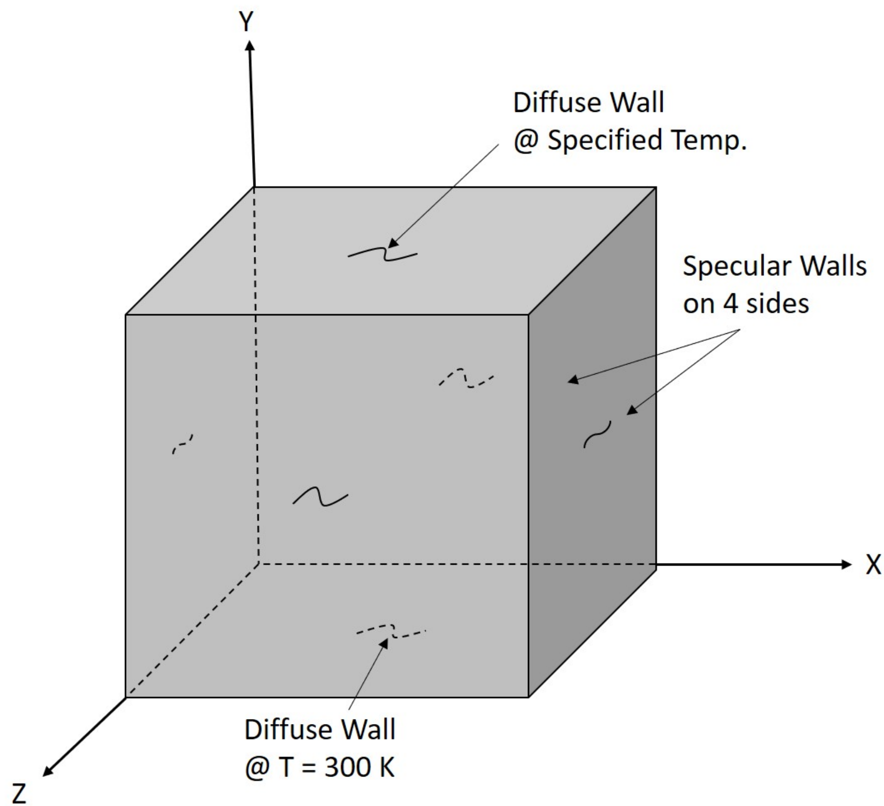


Figure 5.12 – Simulation setup for collisionless heat transfer test case.

Since both infinite parallel plates are simply diffuse walls at specified temperatures, the particles that reflect from those walls are identical to the effusion of molecules from a fictitious equilibrium gas on the other sides of those respective sur-

faces. This means that there is a relationship between the number of particle collisions with each surface and the number density of the fictitious gas at each surface.

In summary, by simply tallying the amount of collisions between particles and both walls, the net heat flux can be computed using Equation 5.3 below:

$$q_f = -2^{3/2}\rho\pi^{-1/2}R^{3/2}T_U^{1/2}T_L^{1/2}(T_U^{1/2} - T_L^{1/2}) \tag{5.3}$$

where ρ is simply found by taking the number of collisions in the simulation with the lower plate, n_L, and using the relation given by Bird that $\rho = 2n_L m$.

Then using the continuum net heat transfer solution of:

$$q_c = \frac{-C(T_U^{\omega+1} - T_L^{\omega+1})}{(\omega+1)h} \tag{5.4}$$

Bird proves the statement that the ratio of the continuum to the free molecule heat transfer is proportional to the Knudsen number of the flow, represented mathematically in Equation 5.5 below.

$$\frac{q_c}{q_f} \propto Kn \tag{5.5}$$

The constant of proportionality is not given nor is it necessarily physically meaningful. The plot of the data can be seen below in Figure 5.13.

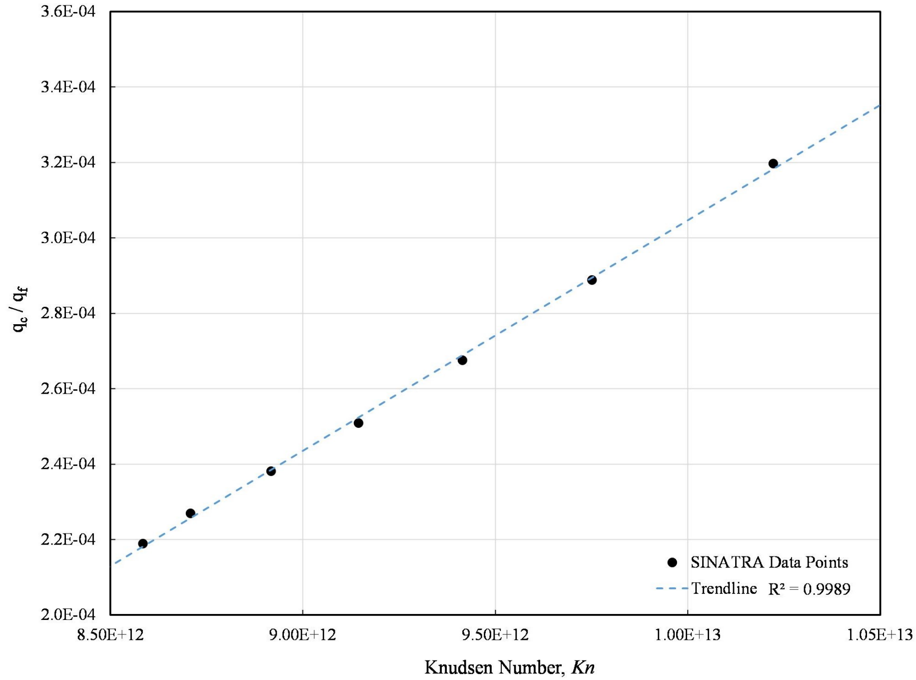


Figure 5.13 – Plot of Collisionless Heat Transfer.

Notice that the R^2 value for the linear fit is 0.9989 which shows that the ratio q_c/q_f is indeed proportional to the Knudsen number. This shows the diffuse walls perform as they are supposed to in SINATRA.

5.5 Molecular Diffusion

It is important to test how particles diffuse with one another to validate collision models. The primary property that indicates the effectiveness of diffusion is the diffusion coefficient as experimentally measured by Chapman and Cowling [43].

5.5.1 Self-Diffusion Coefficient of Argon

The primary test to show how the particles diffuse with one another is simulating the self-diffusion of Argon. This was simulated by having two stationary inflows at opposite ends of the domain, both of different "species" but with the same properties

as Argon. This layout is shown below in Figure 5.14 and the input file used to run the simulation is shown in Appendix E.3. The simulation was run on both a 4096 cell and 32768 cell grid, however the results that were plotted are results from the 32768 cell grid. The results took approximately two weeks to reach a steady state. These long run times are typical for these larger, system-level test cases.

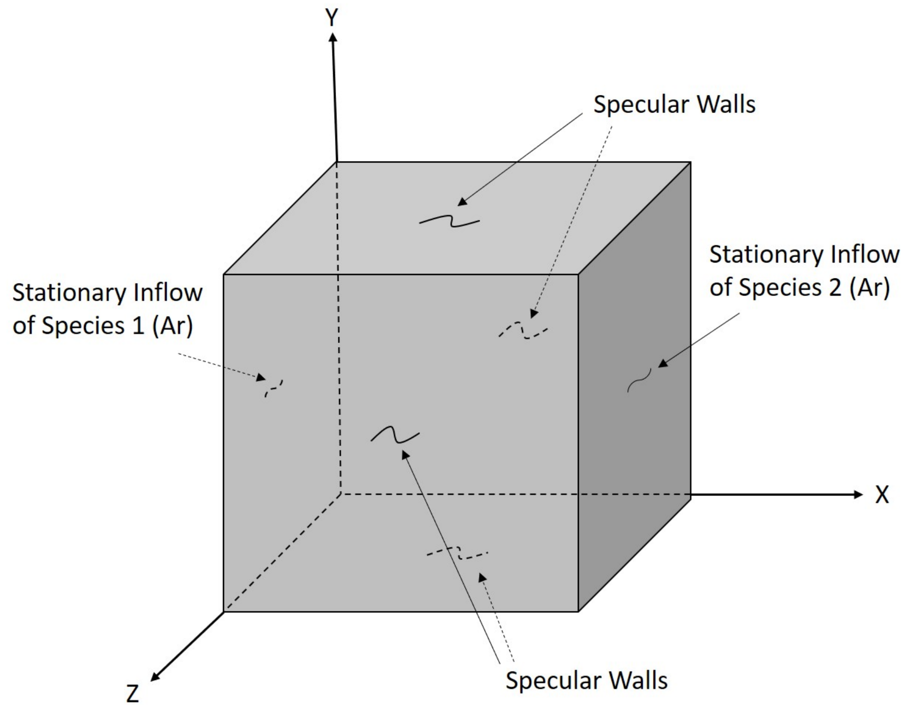


Figure 5.14 – Boundary conditions used to simulate the self-diffusion of Ar.

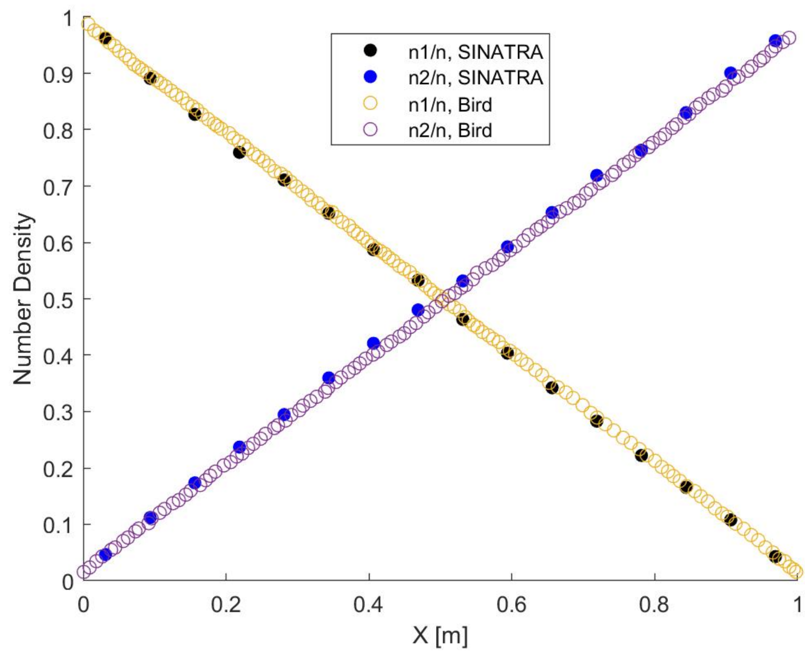


Figure 5.15 – Variation of number densities for both species along the x-axis of the self-diffusion of Ar. Bird's data was extracted from Figure 12.9 in [11].

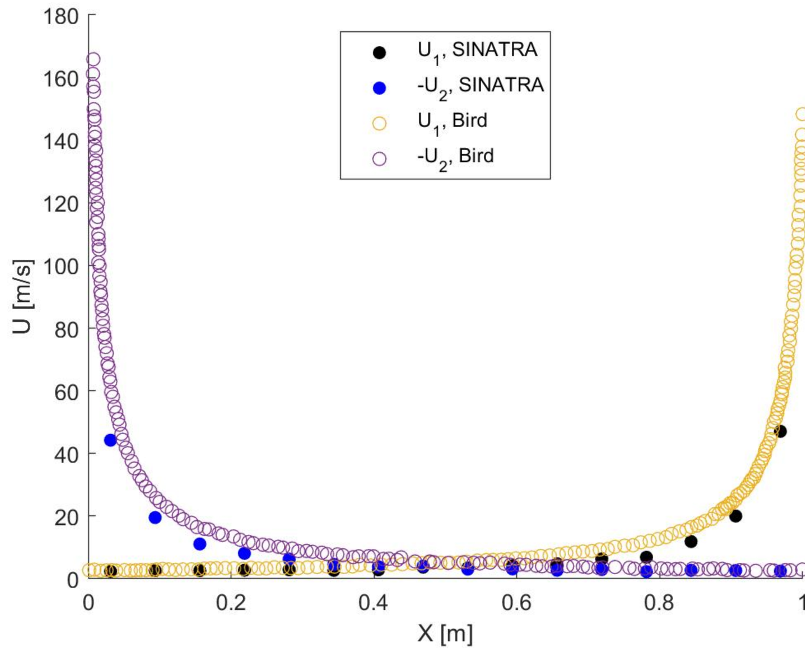


Figure 5.16 – Variation of diffusion velocity along the x-axis of the self-diffusion of Ar. Bird's data was extracted from Figure 12.10 in [11].

The number density plots appear very similar to those of Bird's, however the diffusion velocity plots appear to be off at the endpoints. This may be due to the lack of refinement of the grid close to the walls.

The diffusion coefficient is calculated by Equation 5.6 and the results are summarized below in Table 5.8.

$$D_{11} = D_{12} = -(U_1 - U_2)\frac{n_1 n_2}{n^2}\frac{\Delta x}{\Delta(n_1/n)} \tag{5.6}$$

Table 5.8 – Diffusion coefficient results in comparison to Bird's simulations and theoretical results.

SINATRA Value (m^2s^{-1})	Bird Value (m^2s^{-1})	Percent Error from Bird (%)	Theoretical Value (m^2s^{-1})	Percent Error from Theory (%)
1.08 x 10^{-5}	1.26 x 10^{-5}	14.3	1.57 x 10^{-5}	31

The results don't appear that good especially when compared to theory. However, this is one of the biggest shortfalls of using the VHS model, as it does not accurately capture the diffusion velocities. This is one of the reasons that the VSS model was created, which is an improvement that should be made to SINATRA as discussed in Section 7.3.

5.6 Couette Flow

Couette flow is the flow of a viscous fluid in the space between two surfaces, one of which is moving tangentially relative to the other. Since this is one of the few types of flows that can be solved analytically, it is often used to validate computational algorithms.

A diagram illustrating the boundary conditions applied to the domain are shown below in Figure 5.17.

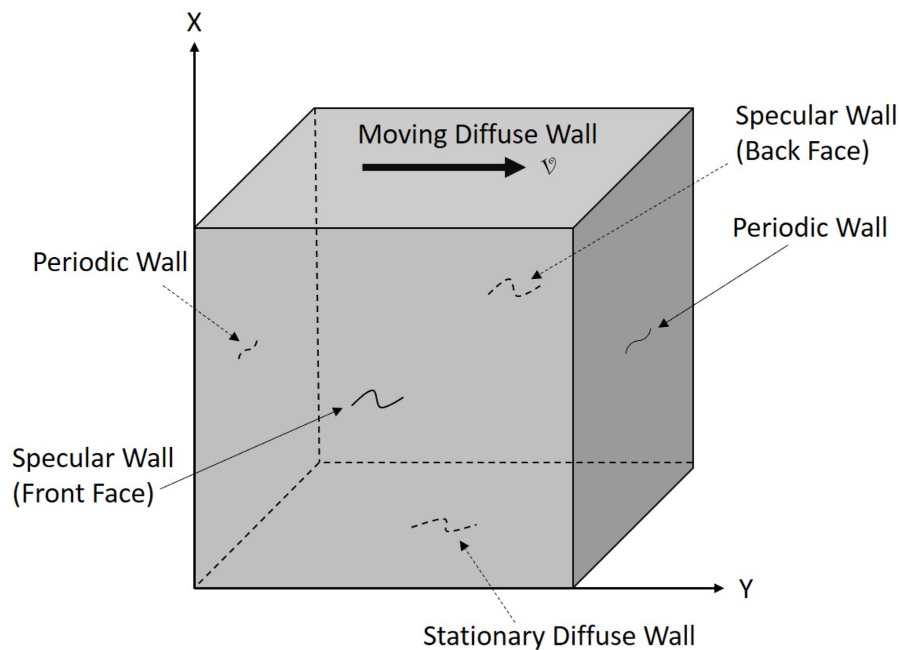


Figure 5.17 – Boundary conditions used to simulate Couette flow (z-axis not shown to limit cluttering).

The top and bottom boundaries are supposed to simulate infinite plates moving parallel to each other. This is accomplished by specifying a velocity, v, on the top plate. The periodic wall is used to simulate an infinite medium. As the flow exits one end, it immediately re-enters at the other end as if the same activity was occurring in that region. Finally, the specular walls act as symmetry planes so that it does not further disturb the flow. The input file used for the Couette flow simulations can be found in Appendix E.4. Note that the only thing needed to change from each simulation is the velocity applied at the top plate (at BC 1) and the species specified (either Argon or Nitrogen).

5.6.1 Viscosity Coefficient of Argon

These test cases attempt to mimic those in [11] Section 12.2 and are actually compared to data extracted from that textbook.

To start off the analysis of Couette flow, a "low speed" wall ($v = 300$ m/s) was applied as the boundary condition for the high x-direction boundary moving in the +y-direction. This simulation was run on a grid containing 4096 cells as shown in Figure 3.10c. However, since the discrete sub-cell scheme was used, there is only a sample resolution equivalent to 512 cells (shown in Figure 3.10b). This only gives eight data points when viewing the three-dimensional results in one dimension. This simulation took approximately six days to complete when run on one processor on Bishop. The exact same simulation was attempted on the next possible refined grid of 32768 cells (Figure 3.10d. However, the dedicated computer time specified for this simulation was 672 hours (or 28 days) and the simulation did not reach steady state.

The velocity gradient is shown below in Figure 5.18, the temperature distribution is shown in Figure 5.19, and the density variation is shown in Figure 5.20. The data from SINATRA is overlayed on data extracted from Bird.

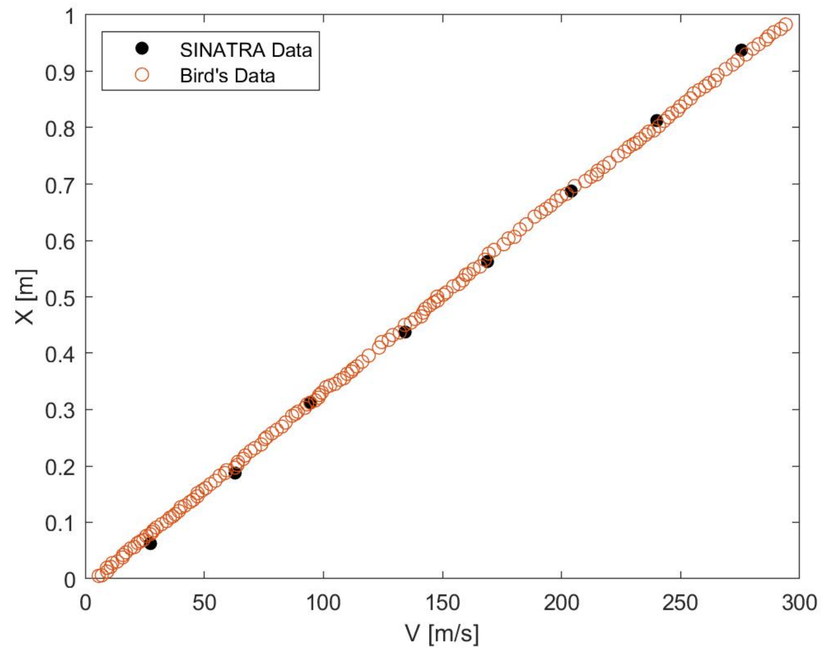


Figure 5.18 – Velocity variation along the x-axis for low speed Couette flow of Ar. Bird's data was extracted from Figure 12.1 in [11].

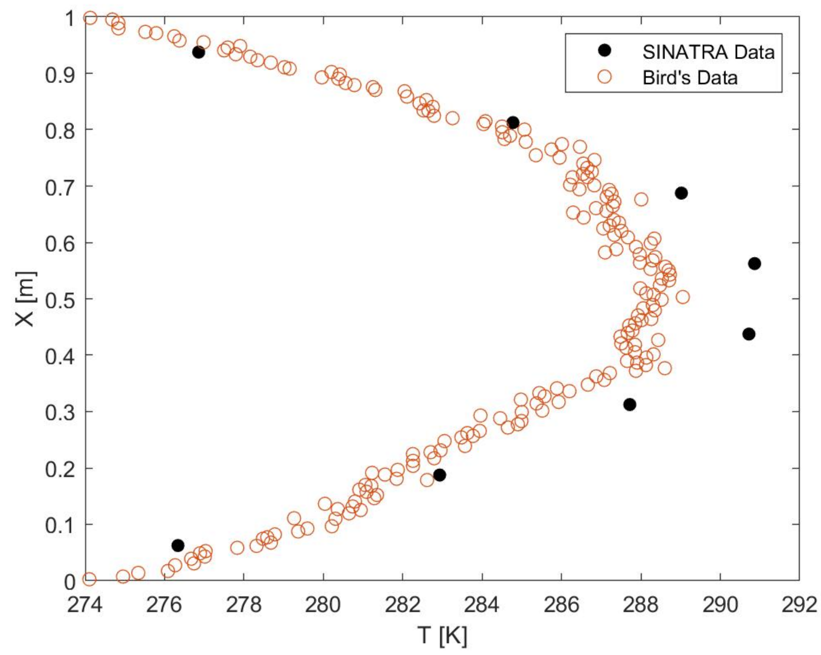


Figure 5.19 – Temperature variation along the x-axis for low speed Couette flow of Ar. Bird's data was extracted from Figure 12.2 in [11].

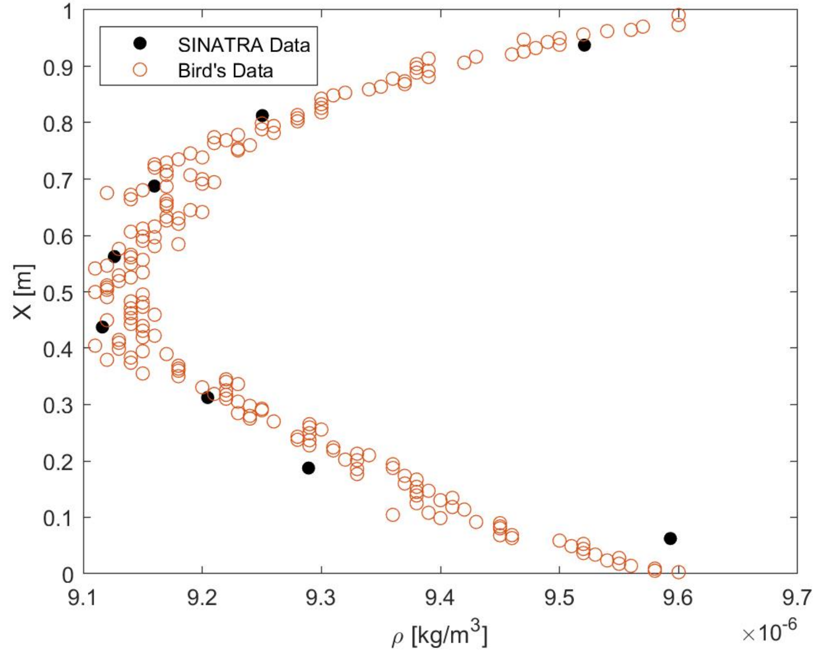


Figure 5.20 – Density variation along the x-axis for low speed Couette flow of Ar. Bird's data was extracted from Figure 12.3 in [11].

The shear stress, τ_{xy} is defined mathematically as:

$$\tau_{xy} = \mu \frac{\partial u}{\partial y} = \mu \frac{\partial v}{\partial x} \tag{5.7}$$

So rearranging this equation and assuming linear variation in velocity along the x-direction, the viscosity coefficient can be found by:

$$\mu = \tau_{xy,avg} \frac{\Delta x}{\Delta v} \tag{5.8}$$

where $\tau_{xy,avg}$ is sampled directly from the flow. The result is shown below in Table 5.9:

Table 5.9 – Viscosity Coefficient result for low speed Couette flow of Ar.

Property	SINATRA Value	Actual Value	Percent Error (%)
μ_{ref} (Nsm^{-2})	1.894 x 10^{-5}	2.117 x 10^{-5}	10.6

Although a 10.6% error is rather large, Bird mentions that it is easier to obtain accurate results for large disturbance flows than small disturbance flows. Due to some of the scatter associated with the low speed flow, the same Couette flow test case was ran with the wall velocity increased to 1000 m/s. This simulation took 549,002 seconds (or 6.35 days) when run on a 4096 cell grid. The same simulation was attempted on a 32768 cell grid but did not reach steady state after the computation time reached its pre-allocated time of 696 hours (or 29 days).

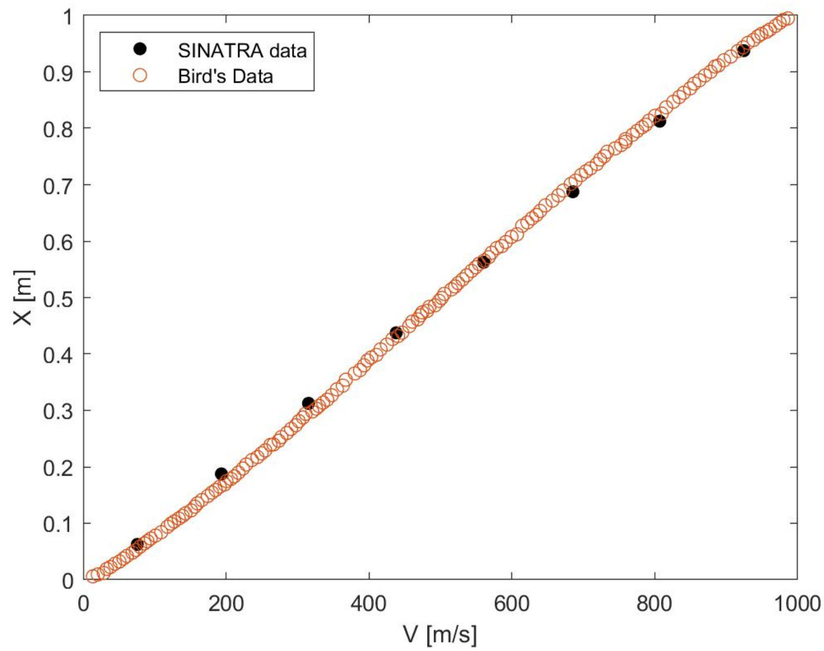


Figure 5.21 – Velocity gradient along the x-axis for high speed Couette flow of Ar. Bird's data was extracted from Figure 12.7 in [11]

The equation used to find the reference value of the coefficient of heat conduction is:

$$K_{ref} = q_x \frac{\Delta x}{\Delta T} \left(\frac{T_{ref}}{T} \right)^{\omega} \tag{5.9}$$

Similar to Bird's methodology for computing the coefficient of heat conduction,

data was only taken from the first three data points closest to the stationary wall (or about the outer 30% of the flow on on that end, in Bird's terms).

The results and their percent errors are shown below in Table 5.10

Table 5.10 – Transport property results for high speed Couette flow of Ar.

Property	SINATRA Value	Actual Value	Percent Error (%)
μ_{ref} (Nsm^{-2})	1.956 x 10^{-5}	2.117 x 10^{-5}	7.6
K_{ref} (Wm^{-1}K^{-1})	0.0160	0.0164	2.2

The results for the coefficients of viscosity and heat conduction are in relatively good agreement with the theory.

5.6.2 Prandtl Number of Nitrogen

The Couette flow test case was also used to validate the Prandtl number of Nitrogen, N_2. This simulation used the same setup as the Argon test cases shown above in Figure 5.17 with a moving wall velocity of 1000 m/s. This simulation was run on a 4096 cell grid and took 594,510 seconds (or 6.88 days) to complete. A simulation on a grid with the next possible level of refinement (32768 cells) was attempted, but did not complete after the maximum allocated time of 696 hours (or 29 days) was met on the Bishop cluster.

The velocity gradient is shown below in Figure 5.22. Bird did not display plots for this flow field in [11], so there is no benchmark data to overlay with SINATRA's data.

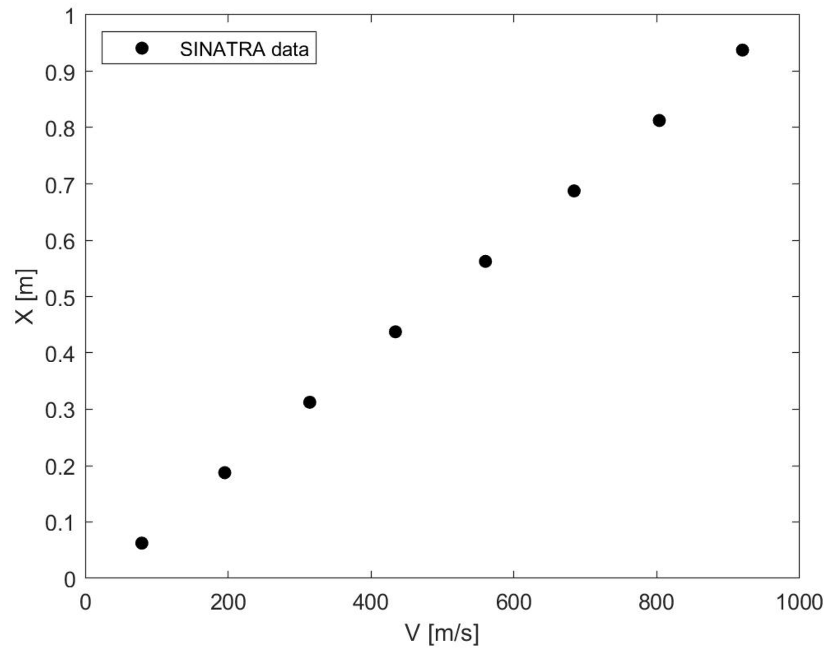


Figure 5.22 – Velocity gradient along the x-axis for high speed Couette flow of N_2.

The Prandtl number is computed using the equation:

$$Pr = \frac{\mu c_p}{K} \tag{5.10}$$

In this case a value of 1039 J/kg-K was used as the specific heat capacity of Nitrogen. The results are summarized below in Table 5.11.

Table 5.11 – Transport property results for high speed Couette flow of N_2.

Property	SINATRA Value	Actual Value	Percent Error (%)
μ_{ref} (Nsm^{-2})	1.5232 x 10^{-5}	1.656 x 10^{-5}	8.0
K_{ref} (Wm^{-1}K^{-1})	0.0243	0.0240	1.3
Pr (-)	0.651	0.72	9.5

Although the result for K_{ref} is very close to the theoretical value, the fact that the viscosity coefficient was found to be lower than theoretical and the heat conduction

coefficient was found to be higher than theoretic made the secondary calculation of the Prandtl number well below the actual value due to error buildup.

Note that generally, the Prandtl number for diatomic gases approximates to 3/4 while the Prandtl number of monatomic gases approximates to 2/3. In reality, one of the primary differences between the two types of molecules are the energy levels associated with each molecule. Since the collisions performed in SINATRA do not yet allow for the exchange of energy (SINATRA is not yet compatible with inelastic collisions) it may make sense for the Prandtl number of Nitrogen to be closer to the general monatomic approximation.

CHAPTER 6

Conclusions

The goal of this thesis project was to develop a DSMC code from scratch and demonstrate successful flow simulations in the hope that the code would later be expanded upon for more complex flows. The structure of the code, SINATRA (SImulatioN of rarefied gases in the upper ATmospheRe And potentially plasma plumes), was outlined and important implementation details were discussed in order to explain to future students how SINATRA operates and give insight into the DSMC method as a whole.

SINATRA was tested against a variety of independent test cases which sought to validate individual aspects of the code. The initialization, boundary conditions, and collisonless flow test cases revealed successful implementation of the basic DSMC procedures for gas flows. However, the collision test cases revealed that better collision schemes and molecular models need to be used in order to better capture the physics of gas flows. It must be understood that the routines implemented in SINATRA are those that were published in the 1990s. Therefore, SINATRA serves as an adequate foundation for further exploration into the DSMC method. Thus, given the nature of SINATRA's current state, the errors quantified in the Results section were expected.

The biggest limiting factor to these simulations was computation time. This has long been a complaint about the DSMC method in general, but there are several ways that can combat this. The following chapter details the specific areas of improvement for SINATRA.

CHAPTER 7

Future Work

Needless to say, there is much room for improvement within the code and even more room for growth in vast sea of opportunity associated with the DSMC method. The rather steep learning curve of programming coupled with the exponential increase in understanding of the DSMC method itself leaves the author wondering where the code would be with another year of work. However, life moves on and there is great hope in confidence in the future students that will develop the code.

That being said, there are several key areas of improvement that should be considered when moving forward with the code. Some tasks are direct and can be accomplished in a relativley short amount of time, while others are more farsighted and will be accomplished through other thesis projects.

There are several routes that can be followed for the future development of the code. Section 7.1 suggests quick modifications to the current code that can produce better results with fewer particles. Sections 7.2, 7.3, and 7.4 explain additions to the current code that can vastly expand the capabilities of the code and certainly make SINATRA results more realistic. These concepts can be explored in great depth, and two master's theses are planned to explore these facets. Sections 7.5 and 7.6 describe opportunities to speed simulation time, but require more computer science knowledge and long-term time commitment. The final sections discuss capabilities that should be added in order to make SINATRA a fully-functional DSMC code.

7.1 Initialization

Although statistical scatter is inherent in the DSMC method, the randomness is captured in the fundamental equations themselves and unnecessary scatter should be avoided when possible. Bird mentions that one key area that can avoid unnecessary scatter is in the initialization process. In SINATRA's current state, the initial position of every particle is assigned randomly throughout the domain as shown in Equation 4.3. However, Bird explicitly states that it would be better to assign the position of every particle *uniformly* rather than *randomly.*

In SINATRA, the initial velocity of every particle is assigned using Equation 4.5. Although this is the correct equation, and the results are satisfactory as shown in Section 5.2.2, it does not guarantee a conservation of momentum. In order to counteract this and guarantee that momentum is conserved in the system, Bird suggests to use "alternative pairs" of particles with exactly opposite velocity components about the average.

Another possible source of excessive statistical scatter in SINATRA's simulations is the way in which particles are injected through inflow and outflow boundaries. In SINATRA's current state, particles are injected into the domain as shown in Section 4.4.1, which spaces the particles on a probabilistic manner based on their velocities, where higher velocites allow for a larger range of possible positions to place the particles. According to Bird, one way to limit the excessive scatter is to force particles to enter boundaries evenly spaced.

Though these improvements were mentioned by Bird himself, the option to either initialize randomly or uniformly should be left to the user and specified in the input file.

7.2 Mesh

One of the key simplifications to the structure of the code is that it is only designed to work with mesh files created by a self-written simple program. Unfortunately there are key shortfalls to the available meshes that make it unreasonable to use for complex flows including local refinement around objects in the domain. Thus, a commercial program needs to be used to create the mesh files and SINATRA needs to properly read in that specific style of file and integrate the information into the existing `Mesh` class.

It is unknown how difficult it will be to integrate these procedures into the code as the developers have not had access to the files outputted by commercial programs. It was the initial objective to use the Cubes meshing software from Cart3D to create the mesh files due to its Cartesian grid style, and it still remains the objective to use this software once licensing details are resolved. Fortunately, the developers of Cart3D offer C++ libraries to help convert the native output from Cubes into familiar styles such as the standard finite element layout, which the current mesh files use. The extra work will lie in processing the other information available from mesh files (such as surface and cut-cell data) which will take extra work to incorporate.

Although compatibility with a commercial meshing program is crucial for more complex simulations, there is still value in working with the simple grids used in this thesis in order to focus more attention on the physics within the cells rather than gas-surface interactions involved in more complex flows. It would be beneficial for a user or developer to work with both types of grids, so SINATRA should maintain the ability to read the simple grids even when more complex grids become available.

7.3 Collisions

There are many different collision models to be used. Certainly different applications require different models, but there is potentially an entire other thesis in itself exploring the different types of collisions and how they affect the results. Recall that there are two critical areas in DSMC that affect how particles collide: molecular models and collision schemes.

Molecular models actually help to make the simulation properties closer to reality by giving realistic transport behavior. The next logical molecular sphere model to implement is the Variable Soft Sphere model (VSS) which would drastically improve the diffusion results. However, more advanced models have come out that should be explored including the Generalized Hard Sphere (GHS) model and the Quantum-Kinetic model.

Also, there are numerous other collison schemes that can be employed that are much more accurate than the NTC method used in SINATRA. These schemes include the nearest neighbor scheme and the trajectory scheme. The in-depth study of these and their effects on the simultion results in itself could be a good scope for a thesis project.

7.4 Energy Considerations

SINATRA currently uses a very simple model for energy that should be improved for more accurate results. However, more than this, there are varying levels of complexity that can be explored in depth, and there are potential thesis projects in themselves exploring how energy models affect DSMC results.

The most immediate aspect of energy that should be incorporated is a more complex vibrational energy scheme. Currently, SINATRA only stores the vibrational

energy from one vibrational energy mode. This is fine as diatomic molecules only have two vibrational degrees of freedom (one vibrational energy mode) so there is only one value of vibrational energy possible. However, the particle data structure needs to be changed in order to be compatible with species of multiple vibrational energy modes.

Next, the simulation should involve inelastic collisions where there is an exchange of internal energy and potentially an energy loss after impact.

It is a key necessity in high speed/high temperature gas simulations to have the particles take part in chemical reactions. Some chemical reactions that arise in these types of flows are dissociations, recombinations, and ionizations.

Finally, further down the road, it would be beneficial to incorporate particle charges and have particles travel through electromagnetic fields. This would be a critical feature necessary for the simulation of plasmas, especially in low earth orbit (CubeSat thrusters). SINATRA's current state is far from this particular ability, although this would be the ultimate goal and key application of this type of code.

7.5 C++: STL versus Pointers

As it stands, SINATRA was written entirely in C++, which is a lower level language than alternatives such as Python or Java. As such, there are considerations to be made when choosing a lower level language: downsides being pickier syntax and more bookkeeping, and the upside being faster speeds. Unfortunately, full advantage was not made of pointers, as many STL vectors were used, which certainly wastes some of the benefits offered by C++. However, the goal of the thesis was to provide the foundation and outline the structure of the code, and leave it as an exercise for future students to optimize its performance. It is true that STL vectors utilize pointers in

their implementation, but it would be better for the program to have full control over memory allocation.

7.6 Parallelization

Interestingly, there are two possible routes to pursue when parallelizing the code:

- Decompose the computational domain into multiple processors, such that kinematics and collision computations within cells are split up
- Use less particles in each simulation, but run a simulation serially in each processor and compute averages (Bird 2013, p 293)

Preliminary research has revealed references [55] and [36] to be good starting points for parallel implementation related directly to DSMC.

It is difficult to write a code from scratch in parallel and often makes more sense to write the code in serial first, then parallelize it after. SINATRA is at the stage where processes work in serial, however there is still more capability to be added before the serial version becomes comprehensive. One option is to take the code down two paths: a serial version and a parallel version. While one student works on parallelizing SINATRA's current serial state, another student can work on expanding on SINATRA's serial state. This will involve a close collaboration between two students, but can be done through resources such as GitHub and weekly meetings.

7.7 Restart Files

An extremely important improvement to SINATRA would be the capability of producing and reading-in restart files at a user-specified frequency of time steps. A restart file essentially takes a snapshot of the simulation at a moment in time and allows the simulation to continue from that moment by simply reading-in and running

the restart file. This eliminates the requirement for SINATRA to start and complete a simulation on one continuous run, and allows simulations to be broken up into multiple runs whether or not it is intentional. This is especially crucial for very long simulations, as the allotted computer time may run out. In SINATRA's current state, if the computer that SINATRA is running on is interrupted, the simulation must be restarted (and successfully completed) to see the end results. This is particularly risky in larger simulations, as weeks of computation time can potentially be lost.

In reality, the main pieces of data that define the present state of the simulation are the properties stored in the particle data structures (`OneParticle` struct stored in the `particleArray` vector). This includes the species, position, velocity, and energy levels of all particles. Presumably, as long as all the particle data is saved, the exact gas flow can be recreated, and the only things left to be specified are those pieces of data already given in the original input file.

One addition to this may be history data of macroscopic surface properties, after the `Boundary` class is created. However, the details of this are not too clear.

The idea would be that once the time for the restart file creation is hit, the key pieces of data would be printed to the restart file.

To create the restart file:

1. Print current simulation time
2. Print computer current computer time (for possible seeding)
3. Print data of every particle in `particleArray` vector
4. Print simulation data from input file including BCs and output info

To restart the simulation:

1. Reset grid and boundary conditions
2. Read in particle data to recreate `particleArray`
3. Link particles and cells

4. Start simulation loop from simulation time printed in restart file (advection, collision, etc.)

Since the restart file will contain a lot of data, it is important to delete the old restart file as soon as the new one is created, to limit stress on computer storage.

7.8 Local Time Stepping

SINATRA currently uses a fixed time step for all particles and cells throughout the entire simulation. Although this accomplishes the goal of simulating particles and their collisions, this is highly inefficient as not all parts of the domain need to move with that small of a time step. Recall from Section 4.6 that the primary parameters driving the time step are the number density, the cell volume, the ratio of real particles to simulated particles, and maximum cross section speed. These parameters are sometimes drastically different from cell to cell, so not all cells need to move at the same time step. Thus, many DSMC codes employ a local time stepping scheme, where each cell operates on its own time step based on the particle activity inside of it.

Before this can be implemented, consideration must be made toward ensuring there is coupling between sampled properties and stored data (see Section 7.10). Two good references for employing a local time step scheme can be found in [56] and [57]

7.9 Addition of `Boundary` Class

In the current state of SINATRA, flow field properties can be sampled but surface properties cannot. In an effort to ensure that the flow properties were correct, gas-surface interactions were temporarily ignored. Thus, one of the first steps moving forward would be to add a `Boundary` class to assist in this process.

This would allow for variation in inlet flow fields as inlet conditions can vary across a domain surface (where currently the entire face has the same inflow properties).

This is especially crucial for flow around bodies in the flow field as properties such as skin friction and pressure distribution on a surface are important for validation.

7.10 Coupling between Sampled Properties and Stored Data

Currently in SINATRA, each cell is sampled, then those properties are printed out to an output file. A step in the simulation should be added to not only sample and print out the sampled data, but also store this data in the `ChildCell` or `ChildInfo` structs associated with the leaf cells. This step can easily be added after each property is sampled for Tecplot output.

7.11 Simulation Stop Triggers

In SINATRA's current state, the user specifies the total simulated time for which the simulation runs. Unfortunately, this relies on the user's ability to know how many seconds (or fractions of a second) it would take for the flow to develop and potentially arrive at a steady state. It would be beneficial to integrate the ability to detect a steady state to eliminate wasted time caused by simulations running longer than necessary, or more importantly, preventing a simulation from terminating before the flow reaches steady state. The ability to detect a steady state would also allow the user to choose if Tecplot output only occurs when steady state has been reached, which would eliminate the generation of unnecessary output files and slightly reduce simulation times.

One possible way to do this would be to track how the macroscopic properties change over time, and once the properties change less than a user-specified percent-

age (similar to a convergence tolerance in numerical methods), the simulation would effectively reach a "steady state".

7.12 Automatic Unit Level Tests

Now that the code has become so large and complex, it is difficult to be confident that additions to the code do not adversely affect past and present abilities of the code. Throughout the development of SINATRA, several different unit level tests have been created to validate different aspects of the code to ensure that it still worked properly after major changes. Unfortunately, no standard set of unit level tests were created to automatically ensure every aspect of the code works, every time. It would be beneficial to create an automatic diagnostics tool that can be run after major changes. Over time, this would speed up the development process as developers can be more confident in each code addition without re-inventing code checks every time.

7.13 DSMC Learning Curve

Bird makes a point to discuss the amount of time necessary to develop a DSMC code. According to his website, it takes "one man-year if it has the minimal level of integrated data input and graphical output and two years for a program with a 'commercial grade' interface". This is one of the most challenging aspects of working on the DSMC code as a master's thesis, since certain aspects and especially perfection of the code necessitates more time, which if fully performed can compete with the work required for a higher degree. For various reasons, some of the concepts involved with DSMC are initially hard to grasp and even more difficult to theorize how to implement in the most general way possible. For this code to succeed over time, it is necessary to have the current primary developer work closely with a successor to ensure that the DSMC method is at least familiar before he or she begins coding.

Cal Poly would benefit from a Computational Fluid Dynamics (or more generally a Computational Sciences) research group devoted to developing and applying multi-physics codes. A research group like this could potentially integrate the knowledge from majors all around Cal Poly which would benefit both the codes developed and the students participating. This would also allow younger students to be exposed to the knowledge required to develop codes and give the higher-level students the experience needed to succeed in advanced engineering analysis.

BIBLIOGRAPHY

[1] E. Kasper and G. Hall, "Introduction to the finite element method: Theory and applications."

[2] S. Chen, M. Wang, and Z. Xia, "Multiscale fluid mechanics and modeling," *23rd International Congress of Theoretical and Applied Mechanics*, 2014.

[3] G. Bird, "Monte carlo simulation of gas flows," *Ann. Rev. Fluid Mech.*, 1978.

[4] B. Barney, "Introduction to parallel computing." `https://computing.llnl.gov/tutorials/parallel_comp/`.

[5] C. P. A. E. Department, "Hpc concepts." `https://aero.calpoly.edu/technology/high-performance-computing/bishop-hpc-cluster/hpc-concepts/`.

[6] A. Garcia, "Traditional dsmc." Presentation.

[7] de Podesta, M., *Understanding Properties of Matter.* CRC Press, 2 ed., 2002.

[8] J. Anderson, *Hypersonic and high-temperature gas dynamics.* American Institute of Aeronautics and Astronautics, Inc., 2 ed., 2006.

[9] B. Moriya, "C programming - pointers." `http://www.exforsys.com/tutorials/c-language/c-pointers.html`.

[10] K. Terekhov, K. Nikitin, M. Olshanskii, and Y. Vassilevski, "A semi-lagrangian method on dynamically adapted octree meshes," *Russian Journal of Numerical Analysis and Mathematical Modeling*, 2015.

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

[12] R. Pletcher, J. Tannehill, and D. Anderson, *Computational Fluid Mechanics and Heat Transfer.* CRC Press, 3 ed., 2013.

[13] M. Bina, *Charge Transport Models for Reliability Engineering of Semiconductor Devices.* PhD thesis, Vienna University of Technology, 2014.

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

[15] P. Gressman and R. Strain, "Global classical solutions of the boltzmann equation without angular cut-off," *Journal of the American Mathematical Society*, vol. 2, no. 771, 2011.

[16] M. Surh, T. Barbee, and L. Yang, "First principles molecular dynamics of dense plasmas," *Physical Review Letters*, vol. 86, p. 5958, 2001.

[17] C. P. M. E. Department, "Cal poly computing power for student learning gets a high performance boost." `https://me.calpoly.edu/news/`.

[18] C. P. A. E. Department, "Bishop full specifications." `https://aero.calpoly.edu/static/media/uploads/hpc_specs.pdf`.

[19] G. Bird, "Approach to translational equilibrium in a rigid sphere gas," *Physics of Fluids*, 1963.

[20] W. Wagner, "A convergence proof for bird's direct simulation monte carlo method for the boltzmann equation," *Journal of Statistical Physics*, vol. 66, no. 3-4, 1992.

[21] D. Rader, M. Gallis, and J. Torczynski, “Direct simulation monte carlo convergence behavior of the hard-sphere-gas thermal conductivity for fourier heat flow,” *Physics of Fluids*, vol. 18, no. 7, 2006.

[22] M. Gallis, J. Torczynski, D. Rader, and G. Bird, “Convergence behavior of a new dsmc algorithm,” *Journal of Computational Physics*, vol. 228, no. 12, 2009.

[23] A. Karchani and R. Myong, “Convergence analysis of the direct simulation monte carlo based on the physical laws of conservation,” *Computers & Fluids*, vol. 115, no. 98-114, 2015.

[24] J. Moss, G. Bird, and G. Markelov, “Dsmc simulations of hypersonic flows and comparison with experiments,” *AIP Conference Proceedings*, 2005.

[25] G. Bird. `http://www.gab.com.au`.

[26] M. Ivanov, G. Markelov, and S. Gimelshein, “Statistical simulation of reactive rarefied flows: Numerical approach and applications,” [58].

[27] M. S. Ivanov, A. V. Kashkovsky, S. Gimelshein, G. Markelov, A. Alexeenko, Y. Bondar, G. A. Zhukova, S. B. Nikiforov, and P. V. Vaschenkov, “Smile system for 2d/3d dsmc computations,” pp. 21–28, 07 2006.

[28] T. Bartel, S. Plimpton, and M. Gallis, *Icarus: A 2-D Direct Simulation Mnte Carlo (DSMC) Code for Multi-Processor Computers*, October 2001.

[29] G. LeBeau and F. Lumpkin II, “Application highlights of the dsmc analysis code (dac) software for simulating rarefied flows,” *Computer Methods in Applied Mechanics and Engineering*, vol. 191, pp. 595–609, 2001.

[30] S. Plimpton and M. Gallis, *SPARTA User’s Manual*, April 2018.

[31] A. Klothakis and I. Nikolos, “Modeling of rarefied hypersonic flows using the massively parallel dsmc kernel ”sparta”,” 07 2015.

[32] D. Liechty, "Object-oriented/data-oriented design of a direct simulation monte carlo algorithm," *11th AIAA/ASME Joint Thermophysics and Heat Transfer Conference*, 2014.

[33] C. White, M. Borg, T. Scanlon, S. Longshaw, B. John, D. Emerson, and J. Reese, "dsmcfoam+: An openfoam based direct simulation monte carlo solver," *Computer Physics Communications*, vol. 224, pp. 22–43, 2018.

[34] T. Scanlon, E. Roohi, C. White, M. Darbandi, and J. Reese, "An open source, parallel dsmc code for rarefied gas flows in arbitrary geometries," *Computers and Fluids*, vol. 39, no. 10, pp. 2078–2089, 2010.

[35] S. Dietrich and I. Boyd, "Scalar and parallel optimized implementation of the direct simulation monte carlo method," *Journal of Computational Physics*, vol. 126, no. 2, pp. 328–342, 1996.

[36] D. Gao and T. Schwartzentruber, "Optimizations and openmp implementation for the direct simulation monte carlo method," *Computers and Fluids*, vol. 42, no. 1, pp. 73–81, 2011.

[37] T. Schwartzentruber and I. Boyd, "A hybrid particle-continuum method applied to shock waves," *Journal of Computational Physics*, vol. 215, no. 2, pp. 402–416, 2006.

[38] T. Schwartzentruber, L. Scalabrin, and I. Boyd, "Multiscale particle-continuum simulations of hypersonic flow over a planetary probe," *Journal of Spacecraft and Rockets*, vol. 45, no. 6, pp. 1196–1206, 2008.

[39] R. Kumar and A. Chinnappan, "Development of a multi-species, parallel, 3d direct simulation monte-carlo solver for rarefied gas flows," *Computers and Fluids*, vol. 159, pp. 204–216, 2017.

[40] B. P. Saponas, "Direct Simulation Monte Carlo on Unstructured Cartesian Grids," Master's thesis, California Polytechnic State University, San Luis Obispo, 2008.

[41] F. White, *Viscous Fluid Flow*. McGraw Hill Education, 3 ed., 2005.

[42] B. Terhal and P. Horodecki, "Schmidt number for ensity matrices," *Physical Review*, 2000.

[43] S. Chapman and T. Cowling, *The mathematical theory of non-uniform gases*. Cambridge University Press, 3 ed., 1970.

[44] G. Bird, *The DSMC Method, Version 1.2*. CreateSpace, 2013.

[45] K. Normak, "Functional programming in scheme with web programming examples," January 2014.

[46] W. Savitch, *Absolute C++*. Addison-Wesley Publishing Company, Inc., 5 ed., 2013.

[47] G. Leavens, "Major programming paradigms." `http://www.eecs.ucf.edu/~leavens/ComS541Fall97/hw-pages/paradigms/major.html#imperative`.

[48] T. Janssen, "Oop concepts for beginners: What is polymorphism." `https://stackify.com/oop-concept-polymorphism/`.

[49] J. Barton and L. Nackman, *Scientific and Engineering C++: An Introduction with Advanced Techniques and Examples*. Addison-Wesley Publishing Company, Inc., 1995.

[50] D. Feszty and T. Jakubik, "8. grid generation." Lecture Notes.

[51] H. Samet, "An overview of quadtrees, octrees, and related hierarchical data structures," *Theoretical Foundations of Computer Graphics and CAD*, 1988.

[52] M. Giles, "Approximating the erfinv function,"

[53] Tecplot, Inc., *Tecplot 360 EX Data Format Guide*, 2015.

[54] J. Camberos, R. Greendyke, and L. Lambe, "On direct simulation quasi-monte carlo methods," AIAA, June 2008.

[55] G. LeBeau, "A parallel implementation of the direct simulation monte carlo method," *Computer Methods in Applied Mechanics and Engineering*, vol. 174, no. 3-4, pp. 319–337, 1999.

[56] M. Laux, "Local time stepping with automatic adaptation for the dsmc method," [58].

[57] C. GuoBiao, S. Wei, and H. FengLong, "Theoretical development for dsmc local time stepping technique," *Science China*, vol. 55, no. 10, pp. 2750–2756, 2012.

[58] *7th AIAA/ASME Joint Thermophysics and Heat Transfer Conference*, AIAA, June 1998.

APPENDIX A

Basic Mesh Generator

```
// Code used to create a mesh file that Tecplot will accept
// Input: Name of file and number of elements along one edge
//       NOTE: numElems must be 2^n where n is any integer greater than 0
// Output: Text file of necessary information

#include <iostream>
#include <string>
#include "FEMeshGenerator.h"
#include <fstream>
#include <cmath>

using std::string;
using std::ofstream;
using std::endl;
using std::ios;

void createFEMesh(string FileName, int numElems)
{
        ofstream outStream;

        // Variables
        int totalNodes = (numElems + 1)*(numElems + 1)*(numElems + 1);
        int totalElems = (numElems)*(numElems)*(numElems);

        outStream.open(FileName);

        // Output first line
        outStream << "TITLE = \"FE Nodes - Unit Cube split into ";
        outStream << (numElems*numElems*numElems) << " Quad Cells\"" << endl;

        // Output second line
        outStream << "VARIABLES = \"X\", \"Y\", \"Z\"" << endl;

        // Third line
        outStream << "ZONE NODES=" << totalNodes << ", ELEMENTS=" << totalElems << ", ";
        outStream << "DATAPACKING=POINT, ZONETYPE=FEBRICK" << endl;

        // Start Node Location Section
        double ds = 1.0 / numElems;
        for (double z = 0; z <= 1.0; z += ds)
        {
                for (double y = 0; y <= 1.0; y += ds)
                {
                        for (double x = 0; x <= 1.0; x += ds)
                        {
                                outStream.setf(ios::fixed);
                                outStream.setf(ios::showpoint);
                                outStream.precision(4);
                                outStream.width(5);
                                outStream << x << "\t" << y << "\t" << z << endl;
                        }
                }
        }

        // Start Connectivity List
        for (int w = 0; w <= ((numElems - 1)*(numElems + 1)*(numElems + 1)); w += (numElems+1)*(numElems+1))
        {
                for (int row = 0; row <= ((numElems-1)*(numElems+1)); row += (numElems+1))
                {
                        for (int n = 1; n <= numElems; n++)
                        {
                                outStream << (n + row + w) << " ";
                                outStream << (n + row + w + 1) << " ";
                                outStream << (n + row + w + 1 + numElems + 1) << " ";
                                outStream << (n + row + w + 1 + numElems) << " ";
                                outStream << (n + row + w + (numElems + 1)*(numElems + 1)) << " ";
                                outStream << (n + row + w + (numElems + 1)*(numElems + 1) + 1) << " ";
                                outStream << (n + row + w + (numElems + 1)*(numElems + 1) + 1 + numElems + 1) << " ";
                                outStream << (n + row + w + (numElems + 1)*(numElems + 1) + 1 + numElems) << endl;
                        }
                }
        }

        outStream.close();
}
```

APPENDIX B

SINATRA Flow Chart and Classes

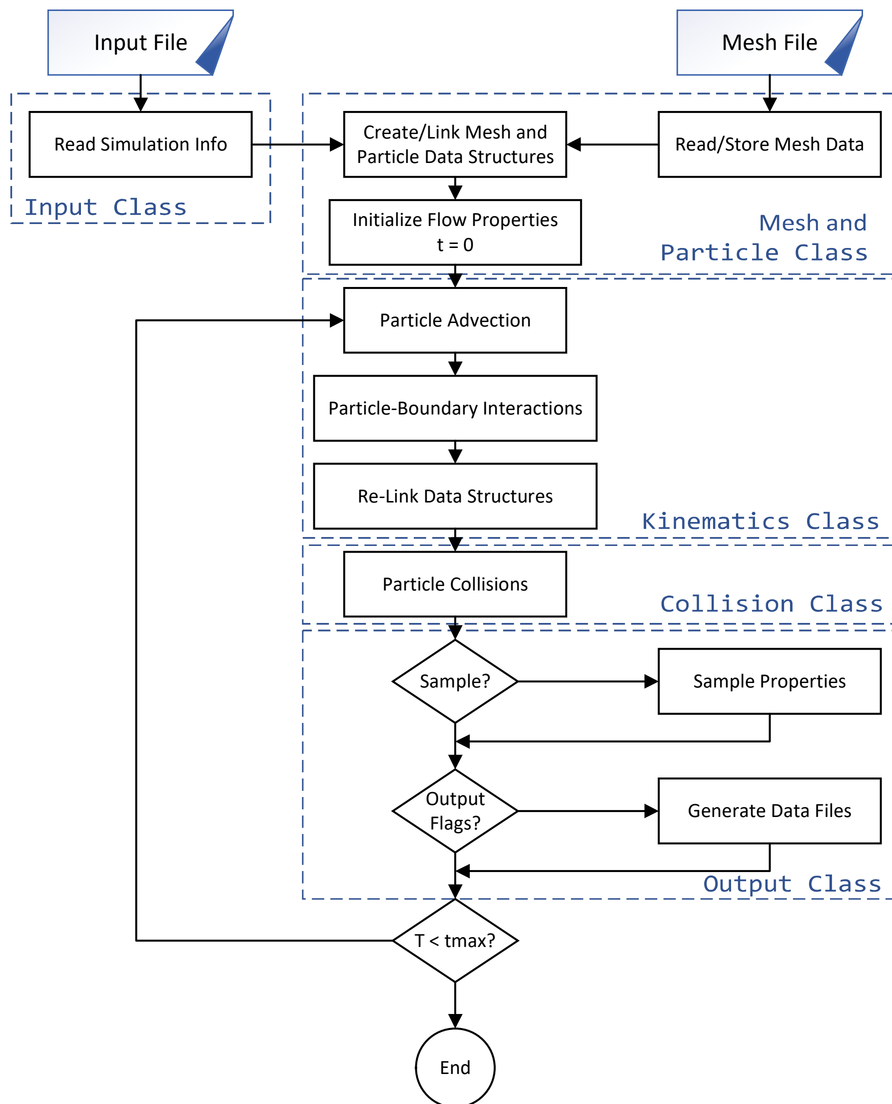


Figure B.1 – SINATRA simulation flow chart with classes that perform each action.

APPENDIX C

SINATRA Code Snippets

C.1 Cell Structures

```
// Structure definitions
struct ChildCell
{
        // ChildCell stores the relatively immediate info required to be a child cell
        int id;   // give some ID value for this child cell
        int indexOfParentCell;  // index of this cell's parent in array parentCells
        int indexOfThisCell;    // index of this cell in array childCells
        int neighbors[6];    // indices of the 6 neighbor cells to this one
        double low_corner[3], high_corner[3]; // stores x,y,z locations of lowest corner and highest corner
        long double maxCrossSpeed; // for collisions
};
struct ChildInfo
{
        // ChildInfo stores important information related to the child cell, but is not as
        // immediately necessary to access
        // Helps break up the storage scheme a little bit
        int numParticles;    // number of particles that lie in the cell
        int firstParticle;   // index of first particle in this cell
                             // this assumes particle structures are stored in a list
                                                // -1 if it has no particles
        double volume;          // physical value of the flow volume in cell

        // structure should also hold info on "weights" associated with the number density in cell
        // used for collision modeling. For now assume that all are the same but info like this should
        // be stored in this structure

        bool isLeaf;            // boolean that marks if the child is a leaf cell
};
struct ParentCell
{
        int id;  // give some ID value for the parent cell
        int indexOfParentCell;  // index of this cell's parent in array parentCells
        int indexOfFirstChild; // index of this parent cell's first child in array childCells
        int level; // level in hierarchical grid
        bool IsGrandparent;  // 1 if this cell is a grandparent, 0 otherwise
        double low_corner[3], high_corner[3]; // stores x,y,z locations of lowest corner and highest corner
};
// STL vectors linking the structs together
vector<ChildCell*> childCells;
vector<ChildInfo*> childInfo;
vector<ParentCell*> parentCells;
```

C.2 Property Sampling

To sample velocities:

```
// Using Bird 2013 Eqn (4.37)
void output::getVelocities(Mesh dsmcMesh, Particle part, int cIdx, int macroCellCount)
{
        // Set up the helper vectors for cell velocities
        // Start by clearing them from any previous values
        cell_xvel.clear();
        cell_yvel.clear();
        cell_zvel.clear();

        cell_xvel.resize(numberOfSpecies);
        cell_yvel.resize(numberOfSpecies);
        cell_zvel.resize(numberOfSpecies);

        // Start loop through particles in this cell
        int firstParticle = (dsmcMesh.childInfo[cIdx])->firstParticle;
        int lastParticle = firstParticle;

        int totalParticles = (dsmcMesh.childInfo[cIdx])->numParticles;

        while (true)
        {
                int isp = (part.particleArray[lastParticle])->indexOfSpecies;

                cell_xvel[isp] += (part.particleArray[lastParticle])->velocity[0];
                cell_yvel[isp] += (part.particleArray[lastParticle])->velocity[1];
                cell_zvel[isp] += (part.particleArray[lastParticle])->velocity[2];

                // End matter to keep loop through particles in macrocells going
                int nextParticleIdx = (part.particleArray[lastParticle])->indexOfNextParticle;
                if (nextParticleIdx < 0)
                        break;
                else
                        lastParticle = nextParticleIdx;
        } // end while loop through particles in macrocells

        double unot_num = 0.0;
        double vnot_num = 0.0;
        double wnot_num = 0.0;
        double unot_denom = 0.0;
        double vnot_denom = 0.0;
        double wnot_denom = 0.0;

        // Iterate through each species
        for (int p = 0; p < numberOfSpecies; p++)
        {
                unot_denom += (part.speciesArray[p])->mass * Np_master[macroCellCount][p];
                vnot_denom += (part.speciesArray[p])->mass * Np_master[macroCellCount][p];
                wnot_denom += (part.speciesArray[p])->mass * Np_master[macroCellCount][p];

                double WEIGHT = ((double)(Np_master[macroCellCount][p]));
                if (WEIGHT == 0.0)
                {
                        unot_num += 0.0;
                        vnot_num += 0.0;
                        wnot_num += 0.0;
                }
                else
                {
                        unot_num += (part.speciesArray[p])->mass * cell_xvel[p]; // weighted velocity here
                        vnot_num += (part.speciesArray[p])->mass * cell_yvel[p];
                        wnot_num += (part.speciesArray[p])->mass * cell_zvel[p];
                }
        }

        unot_vector[macroCellCount] = unot_num / unot_denom;
        vnot_vector[macroCellCount] = vnot_num / vnot_denom;
        wnot_vector[macroCellCount] = wnot_num / wnot_denom;

        cell_xvel.clear();
        cell_yvel.clear();
        cell_zvel.clear();
        return;
}
```

APPENDIX D

MATLAB Particle Animation Script

```
% SINATRA
% Flow Position and Velocity Animation

% David Galvez

clc, clear%, close all

%% Pick what to animate
tog = 5;
anim = 1;
buttonActivate = false;
sve = 0;

%% Data processing
folder = "C:\Users\david\Documents\DAVID\School\Thesis\Code\Development7\resources\VelocityOutput\";

numbers = [1:1:6000];

figure(1)

avgVel = zeros(1, length(numbers));

for i = 1:length(numbers)

    % Convert the number to a string
    strNum = num2str( numbers(i) );

    % Make the string the proper length
    if strlength(strNum) < 5
        while strlength(strNum) <= 5
            strNum = "0" + strNum;
        end
    end

    % Create the full file name
    filename = "velocityFromParticles__" + strNum + ".txt";
    fullFileName = folder + filename;

    fileID = fopen(fullFileName); % Open data file

    % scan text file: specify format, skip header lines, specify delimiter
    C = textscan(fileID, '%f %f %f %f %f %f %f','HeaderLines',6,'Delimiter','\t');

    fclose(fileID);                 % Close data file

    [id, px, py, pz, vx, vy, vz] = C{1,:};    % Defines variables stored in array

    % Remove particles that left the domain
    deleteList = [];
    for j = 1:length(px)
        if px(j) == 2.0
            deleteList(length(deleteList)+1) = j;
        end
    end
    n = 0;
    for j = 1:length(deleteList)
       px((deleteList(j)-n),:) = [];
       py((deleteList(j)-n),:) = [];
       pz((deleteList(j)-n),:) = [];
       vx((deleteList(j)-n),:) = [];
       vy((deleteList(j)-n),:) = [];
       vz((deleteList(j)-n),:) = [];
       n = n + 1;
    end

    if anim == 1
        if tog == 1 % x velocity vs x position (great to debug inlet vel's)
            pause(0.0000001)
            scatter(px, vx)
            axis([0 1 -1500 1500])
        elseif tog == 2 % y velocity vs y position
            pause(0.0000001)
            scatter(px, vy)
            axis([0 1 -1500 1500])
        elseif tog == 3 % z velocity vs z position
            pause(0.0000001)
            scatter(pz, vz)
            axis([0 1 -1500 1500])
        elseif tog == 4 % 2D XY Position Plot
            pause(0.0000001)
            scatter(px, py)
            axis([0 1 0 1])
            xlabel('X Position')
            ylabel('Y Position')
        elseif tog == 5 % 3D plot
            pause(0.0000001)
```

```
            scatter3(px,py,pz)
            axis([0 1 0 1 0 1])
            xlabel('X')
            ylabel('Y')
            zlabel('Z')
            % For top view, uncomment below
            view(0, 90)
            if buttonActivate
                waitforbuttonpress;
            end
        elseif tog == 6 % Follow one particle
            whichParticle = 1; % pick which particle to follow
            pause(0.0000001)
            scatter(px(whichParticle),py(whichParticle))
            axis([0 1 0 1])
            xlabel('X')
            ylabel('Y')
        end
    end

    % Rip out average velocity
    avgVel(i) = mean(vx);

end
```

APPENDIX E

Simulation Input Files

E.1 Mesh Convergence

NOTE: FNUM value changes to match desired number of particles per cell

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

Number of Real Particles to Simulation Particles
3.05175781250E+15

Boundary Conditions
DWALL DWALL DWALL DWALL DWALL DWALL

Collision Scheme
0

Sphere Model
2

Total Simulation Time
0.00100

Time Step
0.00001

Initial Conditions

Number Density
1.0e+020

Mixture
1 1.0

Stream Temperature
273.0

Stream Velocity
0.0 0.0 0.0

***

Boundary Condition Information

BC 0

Number Density
1.0e+020

Mixture
1 1.0

Stream Temperature
273.0

Stream Velocity
0.0 0.0 0.0

&

BC 1

Number Density
1.0e+020

Mixture
1 1.0

Stream Temperature
273.0

Stream Velocity
0.0 0.0 0.0

&

BC 2

Number Density
1.0e+020

Mixture
1 1.0

Stream Temperature
273.0

Stream Velocity
0.0 0.0 0.0

&

BC 3
```

```
Number Density
1.0e+020
Mixture
1 1.0
Stream Temperature
273.0
Stream Velocity
0.0 0.0 0.0
&
BC 4
Number Density
1.0e+020
Mixture
1 1.0
Stream Temperature
273.0
Stream Velocity
0.0 0.0 0.0
&
BC 5
Number Density
1.0e+020
Mixture
1 1.0
Stream Temperature
273.0
Stream Velocity
0.0 0.0 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 Start Time
0.00001
Tecplot Sample Frequency
1
Particle Animation Switch
0
Particle Animation Start Time
0.00001
Particle Animation Frequency
1
***
Species Information
Species 1 (Nitrogen, N2)
273
4.17e-010
4.65e-026
2
2
2.88
3371
0.74
1.656e-05

***
Optional Keywords
***
```

E.2 Collisionless Heat Transfer

NOTE: Input file is of older format, but still runs with updated code

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

Number of Real Particles to Simulation Particles
1e+013

Boundary Conditions
SWALL SWALL DWALL DWALL SWALL SWALL

Collision Scheme
0

Sphere Model
1

Total Simulation Time
0.05000

Time Step
0.00001

Initial Conditions

Number Density
1e+017

Mixture
1 1.0

Stream Temperature
300.0

Stream Velocity
0.0 0.0 0.0

***

Boundary Condition Information

BC 2

Number Density
1e+023

Mixture
1 1.0

Stream Temperature
300.0

Stream Velocity
0.0 0.0 0.0

&

BC 3

Number Density
1e+023

Mixture
1 1.0

Stream Temperature
1000.0

Stream Velocity
0.0 0.0 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
100000

***

Species Information

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

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

Species 3 (Argon, Ar)
273
4.17e-010
```

```
6.63e-026
0
0
0
0
0.81
2.117e-05

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

***
```

E.3 Self-Diffusion Flow

```
*** SINATRA Input File ***
Number of Real Particles to Simulation Particles
2.8e+015
Boundary Conditions
INFLOW INFLOW SWALL SWALL SWALL SWALL
Collision Scheme
1
Sphere Model
2
Total Simulation Time
0.70000
Time Step
0.000008
Initial Conditions
Number Density
1.4e+020
Mixture
1 0.5 2 0.5
Stream Temperature
273.0
Stream Velocity
0.0 0.0 0.0
***
Boundary Condition Information
BC 0
Number Density
1.4e+020
Mixture
1 1.0
Stream Temperature
273.0
Stream Velocity
0.0 0.0 0.0
&
BC 1
Number Density
1.4e+020
Mixture
2 1.0
Stream Temperature
273.0
Stream Velocity
0.0 0.0 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 Start Time
0.00001
Tecplot Sample Frequency
25
Particle Animation Switch
0
Particle Animation Start Time
0.00001
Particle Animation Frequency
1
***
Species Information
Species 1 (Argon, Ar)
273
4.17e-010
6.63e-026
0
0
0
0
0.81
2.117e-05
Species 2 (Argon, Ar)
273
4.17e-010
6.63e-026
0
0
0
```

```
0
0.81
2.117e-05
***
Optional Keywords
diffusion
***
```

E.4 Couette Flow

```
*** SINATRA Input File ***
Number of Real Particles to Simulation Particles
2.8e+015
Boundary Conditions
DWALL DWALL PWALL PWALL SWALL SWALL
Collision Scheme
1
Sphere Model
2
Total Simulation Time
0.90000
Time Step
0.00002
Initial Conditions
Number Density
1.4e+020
Mixture
1 1.0
Stream Temperature
273.0
Stream Velocity
0.0 0.0 0.0
***
Boundary Condition Information
BC 0
Number Density
1e+023
Mixture
1 1.0
Stream Temperature
273.0
Stream Velocity
0.0 0.0 0.0
&
BC 1
Number Density
1e+023
Mixture
1 1.0
Stream Temperature
273.0
Stream Velocity
0.0 300.0 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 Start Time
0.40000
Tecplot Sample Frequency
25
Particle Animation Switch
0
Particle Animation Start Time
0.00005
Particle Animation Frequency
1
***
Species Information
Species 1 (Argon, Ar)
273
4.17e-010
6.63e-026
0
0
0
0
0.81
2.117e-05
***
Optional Keywords
***
```

APPENDIX F

Bishop Batch Script

```
#!/bin/bash
#
#SBATCH --ntasks=1
#SBATCH --job-name=TitleOfSim.job
#SBATCH --output=output.%j.out
#SBATCH --time=696:00:00
#SBATCH --partition=extendedq

rm DSMC
g++ -Wall Development7.cpp Mesh.cpp Simulation.cpp particle.cpp input.cpp Kinematics.cpp Collision.cpp output.cpp
 Debugger.cpp  -o DSMC
./DSMC
```