1994

Stochastic modeling of a dilute gas with buffered regions

James Howard Colby
San Jose State University

Follow this and additional works at: https://scholarworks.sjsu.edu/etd_theses

Recommended Citation

Colby, James Howard, "Stochastic modeling of a dilute gas with buffered regions" (1994). *Master's Theses*. 826.

DOI: <https://doi.org/10.31979/etd.75vq-h3g7>
https://scholarworks.sjsu.edu/etd_theses/826

This Thesis is brought to you for free and open access by the Master's Theses and Graduate Research at SJSU ScholarWorks. It has been accepted for inclusion in Master's Theses by an authorized administrator of SJSU ScholarWorks. For more information, please contact scholarworks@sjsu.edu.

INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

U·M·I

University Microfilms International
A Bell & Howell Information Company
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
313/761-4700, 800/521-0600

Order Number 1359009

Stochastic modeling of a dilute gas with buffered regions

Colby, James Howard, M.S.

San Jose State University, 1994

U·M·I

**300 N. Zeeb Rd.
Ann Arbor, MI 48106**

STOCHASTIC MODELING OF A DILUTE GAS
WITH BUFFERED REGIONS

A Thesis
Presented to
The Faculty of the Department of Physics
San Jose State University

In Partial Fulfillment
of the Requirements for the Degree
Master of Science

by
James Howard Colby
August, 1994

© 1994

James Howard Colby

ALL RIGHTS RESERVED

APPROVED FOR THE DEPARTMENT OF PHYSICS

Dr. Alejandro Garcia

Dr. Patrick Hamill

Dr. Francis Alexander

APPROVED FOR THE UNIVERSITY

ABSTRACT

STOCHASTIC MODELING OF A DILUTE GAS WITH BUFFERED REGIONS

By James H. Colby

The effects of using buffered regions to model boundary conditions and the proper calculating of average velocity within the buffered regions were studied using the Direct Simulation Monte Carlo algorithm for a dilute gas. Replacing thermal boundaries with buffered regions that fixed density, velocity, and temperature were examined for several cases: thermodynamic equilibrium; Couette flow; and a system with a temperature gradient. Average velocity was computed using two methods, by averaging the instantaneous velocity, and by averaging the thermodynamic velocity (instantaneous momentum).

Buffered regions modeled boundary conditions very well, except when density was fixed for the temperature gradient system. In this case, the fixed density regions established a small velocity drift across the system. Also demonstrated is the need to compute the average velocity using thermodynamic velocity for the fixed density cases. These difficulties with fixed density regions are important considerations when patching to a non-DSMC modeled region.

TABLE OF CONTENTS

List of Figures	vi
Chapter 1 Introduction	1
Chapter 2 Kinetic Gas Theory	4
Basic Theory	4
Boltzmann's Equation	6
Maxwell-Boltzmann Distribution	7
Mean Free Path and Collision Frequency	9
Flow Parameters	11
Boundary Conditions	13
Simple Dilute Gas Systems	17
Chapter 3 Computational Methods	21
Continuum-Based Analytic Equations	21
Direct Particle Simulations	28
Comparison of Methods	31
Chapter 4 DSMC Simulations	32
Code Description	32
Thermodynamic Equilibrium Simulation	38
Couette Flow Simulation	39
Temperature Gradient Simulation	39
Chapter 5 Simulations Results	41
Thermodynamic Equilibrium Results	42
Couette Flow	51
Temperature Gradient Simulation	64
Chapter 6 Conclusions	79
Bibliography	82
Appendix A DSMC Source Code	84

LIST OF FIGURES

Figure

1.	Maxwell-Boltzmann distribution of particle velocities for argon at $T=273$ K. The vertical line marks the average speed, $\langle v \rangle$	9
2.	Particle moving amid a field of stationary particles.	10
3.	The delineation of regimes for air at standard pressure and temperature as a function of density and characteristic length.	12
4.	Boundary surface imperfections result in two types of collisions - specular and thermal.	14
5.	Velocity (z-component) probability distribution for particles leaving a thermal reservoir.	16
6.	Planar Couette flow. Left and right walls move at constant velocities, but in opposite directions.	17
7.	Schematic for Maxwell's estimate of viscosity.	18
8.	Comparison of temperature profiles for hard sphere thermal diffusivity (solid line) and constant thermal diffusivity (dashed line).	20
9.	Fluid moving through a control volume.	23
10.	Elemental volume for momentum equation. Only stress forces in the x -direction are shown. The y - and z -components are obtained in a similar manner.	25
11.	Alignment of buffer cell and non-buffer cell systems.	40
12.	Average density results of thermodynamic equilibrium case, with and without buffer cells.	43
13.	Average z -component of velocity across system.	44
14.	Instantaneous average velocity and average velocity computed from instantaneous average momentum.	45
15.	Average y -component of velocity across system.	46
16.	Average x -component of velocity in each cell.	47
17.	Average temperature within each cell.	48
18.	Probability density for particles crossing into cell 1.	49
19.	Probability distribution for particles crossing out of cell 1.	50
20.	Probability distribution for particles crossing out of cell 21.	50

LIST OF FIGURES (cont'd)

Figure

21.	Probability distribution for particles crossing into cell 21.	51
22.	Overlay of particle densities for non-buffer and fixed density buffer cell Couette flows.	52
23.	Overlay of temperature distributions for non-buffer and fixed density buffer cell Couette flows.	54
24.	Overlay of v_z distributions for non-buffer and fixed density buffer cell Couette flows.	55
25.	Overlay of v_y distributions for non-buffer and fixed density buffer cell Couette flows.	56
26.	Overlay of v_x distributions for non-buffer and fixed density buffer cell Couette flows.	57
27.	Overlay of average v_z computed using instantaneous and thermodynamic time averaged velocities for Couette flow with fixed density buffer cells.	58
28.	Probability distribution for particles crossing into cell 1.	59
29.	Probability distribution for particles crossing out of cell1.	59
30.	Probability distribution for particles crossing out of cell 21.	60
31.	Probability distribution for particles crossing into cell 21.	60
32.	Overlay of density results for non-buffer cell and variable density buffer cell simulations.	61
33.	Overlay of temperature results for non-buffer cell and variable density buffer cell simulations.	62
34.	Overlay of v_z results for non-buffer cell and variable density buffer cell simulations.	63
35.	Overlay of velocity results for variable density buffer cell simulations.	64
36.	Particle density results for temperature gradient simulations of non-buffer cell and fixed density buffer cell systems.	65
37.	Temperature distributions for temperature gradient simulations of non-buffer cell and fixed density buffer cell systems.	66

LIST OF FIGURES (cont'd)

Figure

38.	The v_z distributions for temperature gradient simulations of non-buffer cell and fixed density buffer cell systems.	67
39.	The v_y distributions for temperature gradient simulations of non-buffer cell and fixed density buffer cell systems.	68
40.	The v_x distributions for temperature gradient simulations of non-buffer cell and fixed density buffer cell systems.	69
41.	Particle densities for non-buffer cell and modified boundary conditions fixed density buffer cell simulations.	70
42.	Overlay of temperature profiles for non-buffer cell and modified boundary conditions fixed density buffer cell simulations.	71
43.	Overlay of v_z profiles for non-buffer cell and modified boundary conditions fixed density buffer cell simulations.	72
44.	Overlay of density profiles for non-buffer cell and variable density buffer cell simulations.	73
45.	Overlay of temperature profiles for non-buffer cell and variable density buffer cell simulations.	74
46.	Overlay of v_z profiles for non-buffer cell and variable density buffer cell simulations.	75
47.	Probability distribution for particles crossing into cell 1.	76
48.	Probability distribution for particles crossing out of cell 1.	77
49.	Probability distribution for particles crossing out of cell 21.	78
50.	Probability distribution for particles crossing into cell 21.	78

CHAPTER 1

INTRODUCTION

Computer simulations of fluid dynamics employ a variety of methods to characterize fluid properties and describe fluid behavior under particular conditions of temperature, pressure, and density. While some simulations solve systems of partial differential equations, others model a fluid by tracking positions, velocities and collision interactions of individual particles comprising the fluid [1].

Particle methods are especially suited to the modeling of dilute gases, where a simplifying assumption further reduces the complexity of the simulation. Molecular Dynamics is one particle method that can be used to model any fluid. Only ordinary differential equations are involved in this method, with the dynamics developed using small, finite time steps. However, simulations are limited to very short times due to the massive computing resources required to keep track of every molecule comprising the fluid, even for dilute gases. By making the assumption that for a dilute gas collisions only involve two particles, not three or more, a probabilistic approach using Boltzmann statistics can replace detailed particle tracking methods like Molecular Dynamics.

A stochastic method that only considers binary collisions (collisions between two particles) is called Discrete Simulation Monte Carlo (DSMC), first developed by Bird in the early 1960's. While DSMC records particle positions in order to determine boundary interactions, collisions between particles are processed by computing the probability of collision based on relative velocities for particles randomly chosen within a given space. Because of the stochastic nature of the collision calculations, each DSMC

particle can represent a large number of molecules in a physical system, as opposed to the one-to-one correspondence between molecules and particles in a Molecular Dynamics model. This allows DSMC to realistically model a macroscopically large system of molecules with a much smaller number of representative particles [2],[3],[4].

One feature of the DSMC method is its ability to control the density, temperature, and velocity of the fluid within a particular space in the simulation volume. This feature allows an alternate method for treating boundary conditions by controlling relevant parameters within the *buffered* regions that are consistent with the boundary conditions. These buffered regions are especially useful for patching a DSMC-modeled space to a space modeled by another method, such as Navier-Stokes [5]. This work examines the use of these buffering regions and the effects on fluid properties such as velocity, temperature, and density in a DSMC simulated region bounded by two buffered regions. The proper calculating of average quantities in the buffered regions will be demonstrated, as well as deviations from expected behavior for certain fixed buffer cell parameters.

Results for three simulations will be presented: thermodynamic equilibrium, Couette flow, and flow resulting from a temperature gradient. Two methods will also be examined for computing average fluid velocities in order to determine the correct one for describing average velocities in the buffering regions.

The results of this work indicate that buffered regions implement boundary conditions very well, but that great care is required if the particle density is fixed. Small density gradients can arise, causing a velocity gradient across the system even when there is no physical reason for the gradient to

exist. Also demonstrated is that when buffered region densities are fixed, the average velocities must be computed using thermodynamic velocity, that is, using time-averaged momentum instead of time-averaged velocity. These difficulties with fixed density buffered regions are important considerations when patching to a non-DSMC modeled region.

CHAPTER 2

KINETIC GAS THEORY

2.1 Basic Theory

The fundamental theory governing fluid dynamics is based on the universal conservation laws of mass, momentum, and energy. From applying these laws to a fluid flow, the continuity equation (conservation of mass), the momentum equation (conservation of momentum), and the energy equation (conservation of energy) are derived. Through the use of these equations plus constituent relationships between fluid properties, a full set of equations modeling a fluid's flow is obtained. An example of such a constituent relationship is the equation of state, which relates the thermodynamic variables of pressure (p), density (ρ), and temperature (T).

For a gas, one approach to developing a system of equations describing flow is the use of integral relations involving the dynamics of colliding particles. The necessary transport coefficients relating stress and rate of strain, and heat flux and temperature are defined in this approach by the collision integrals. Mathematical uncertainty is introduced by using probability theory to evaluate the collision integrals. This approach is called the kinetic theory of gases [6].

Simple gases modeled by the kinetic theory of gases are described by the thermodynamic variables mentioned above along with the volume, V , the gas occupies in space. The gas is considered dilute if the molecular spacing, δ , between particles in the volume (system) is large with respect to the molecules' effective diameter, d_{eff} , for a single species gas,

$$\sqrt[3]{1/n} = \delta \gg d_{\text{eff}} \quad (\text{EQ 1})$$

The number density, n , is the total number of particles per unit volume.

The effective diameter is defined by choosing an experimentally derived or theoretically modeled interaction potential. For example, in the hard sphere model used in this work, the effective diameter is equivalent to the particle's molecular diameter.

Once an interaction potential is chosen, the entire dynamics of the system is given by the equation of motion,

$$\ddot{\mathbf{r}}_i = \frac{1}{m} \sum_{\substack{j=1 \\ j \neq i}}^N \mathbf{F}_{ij} \quad (\text{EQ 2})$$

where m is the mass of a particle, F is the interaction potential, with $F_{ij} = F(\mathbf{r}_i - \mathbf{r}_j) = -F_{ji}$, and $\mathbf{r}_i$ and $\mathbf{r}_j$ are the positions of particles i and j , respectively. The numerical evaluation of this system of ordinary differential equations is called Molecular Dynamics and has been very successful in computing microscopic properties of fluids.

While the numerical evaluation of equation 2 has been successful, it is limited to very small systems due to the number of particles involved. However, another method has been developed in modeling dilute gases, where single collisions are primarily between two particles only, allowing for a probabilistic approach to be used in developing the model. This method is called Discrete Simulation Monte Carlo (DSMC) and is based on Boltzmann's equation.

2.2 Boltzmann's Equation

First proposed by Boltzmann in 1872, the Boltzmann equation describes the dynamics of systems with predominantly binary collisions, i.e., dilute gases. The Boltzmann equation expresses the change in a given statistical velocity distribution, $f(\mathbf{c}, \mathbf{r}, t)$, where $\mathbf{c}$ represents the velocities of the particles, $\mathbf{r}$ their positions, and t is the time, for all time and space.

The three processes composing the Boltzmann equation are: free transport (particle movement), external influences (forces), and particle-particle interactions (collisions). The changes in velocity distributions can be considered independently, and then combined to determine the total change in the velocity distribution $f(\mathbf{c}, \mathbf{r}, t)$.

The change in the distribution due to particle movement can be expressed as the dot product of the velocity class $\mathbf{c}$ (the velocities of the particles), with the spatial gradient of the velocity distribution,

$$\text{free transport} \equiv -\mathbf{c} \cdot \frac{\partial}{\partial \mathbf{r}} (nf) \quad (\text{EQ 3})$$

where n is the number density.

For each external force $\mathbf{F}_i$ acting on the system, the change in the particle distribution can also be expressed as a dot product. In this case, however, the equation is the dot product of the external force with the change in the particle distribution with respect to velocity, $\mathbf{c}$.

$$\text{external influences} \equiv -\sum_i \left(\mathbf{F}_i \cdot \frac{\partial}{\partial \mathbf{c}} (nf) \right) \quad (\text{EQ 4})$$

Finally, the distribution changes due to collisions occurring among particles are calculated. The overall increase in particles of velocity class $\mathbf{c}$, by

the collision of particles in another velocity class c_1 with particles in velocity class c , is expressed in equation 5. Given a post-collision distribution f^* , the first term in the integrand represents the injection of particles into velocity class c , while the second term represents the removal of particles from velocity class c .

$$\text{particle interactions} \equiv \int_{-\infty}^{\infty} \int_0^{4\pi} n^2 (f^* f_1^* - ff_1) (c - c_1) \sigma d\Omega dc_1 \quad (\text{EQ 5})$$

where σ is the collision cross section, and Ω is the solid angle.

The complete Boltzmann equation can therefore be expressed as the sum of these three particle velocity distribution changing processes,

$$\begin{aligned} \frac{\partial}{\partial t}(nf) = & \left(-\mathbf{c} \cdot \frac{\partial}{\partial \mathbf{r}}(nf) \right) - \sum_i \left(\mathbf{F}_i \cdot \frac{\partial}{\partial \mathbf{c}}(nf) \right) \\ & + \int_{-\infty}^{\infty} \int_0^{4\pi} n^2 (f^* f_1^* - ff_1) (c - c_1) \sigma d\Omega dc_1 \end{aligned} \quad (\text{EQ 6})$$

For particles approaching equilibrium, the velocity distribution approaches the Maxwell-Boltzmann distribution. This is also known as Boltzmann's H-theorem [7]. However, the same velocity probability distribution can be derived from the fundamental axiom of the canonical ensemble.

2.3 Maxwell-Boltzmann Distribution

The fundamental axiom of the canonical ensemble [8] is that for a system at thermodynamic equilibrium with temperature T , the probability that a particle is at a position between $\mathbf{r}$ and $\mathbf{r}+d\mathbf{r}$ with a velocity between $\mathbf{v}$ and $\mathbf{v}+d\mathbf{v}$ is,

$$P(\mathbf{r}, \mathbf{v}) d\mathbf{r} d\mathbf{v} = A e^{-E(\mathbf{r}, \mathbf{v})/kT} d\mathbf{r} d\mathbf{v} \quad (\text{EQ 7})$$

where E is the particle's energy and A is a normalization constant.

For a dilute gas, the gas is assumed to be ideal, so that all of a particle's energy is kinetic,

$$E(\mathbf{r}, \mathbf{v}) = \frac{1}{2} m |\mathbf{v}|^2 \quad (\text{EQ 8})$$

This is a good approximation for a dilute gas since particles spend most of their time in free flight, briefly interacting with single particles (binary collisions).

Because all of a particle's energy is assumed to be kinetic, it is independent of $\mathbf{r}$ and the probability density can then be expressed as

$$\begin{aligned} P(\mathbf{r}, \mathbf{v}) d\mathbf{r} d\mathbf{v} &= (P(\mathbf{r}) d\mathbf{r}) (P(\mathbf{v}) d\mathbf{v}) \\ &= \left(\frac{1}{V} \right) d\mathbf{r} (P(\mathbf{v}) d\mathbf{v}) \end{aligned} \quad (\text{EQ 9})$$

Since $\mathbf{r}$ and $\mathbf{v}$ are decoupled in the probability density $P(\mathbf{r}, \mathbf{v})$ when the ideal gas assumption is used, an expression for the velocity probability distribution, $P(\mathbf{v})$ can be obtained from equation 7. Also making use of the fact that $P(\mathbf{r}, \mathbf{v})$ is isotropic, the probability that a particle has a velocity between $\mathbf{v}$ and $\mathbf{v}+d\mathbf{v}$ is

$$\begin{aligned} P(\mathbf{v}) d\mathbf{v} &= P(v, \theta, \phi) v^2 \cos\theta dv d\theta d\phi \\ &= A e^{-\frac{1}{2}mv^2/kT} v^2 dv \cos\theta d\theta d\phi \end{aligned} \quad (\text{EQ 10})$$

Integrating the angular parts, since the distribution of velocities is isotropic, and imposing normalization to determine A gives,

$$P(\mathbf{v}) d\mathbf{v} = 4\pi \left(\frac{m}{2\pi kT} \right)^{3/2} v^2 e^{-\frac{1}{2}mv^2/kT} dv \quad (\text{EQ 11})$$

This velocity distribution is the Maxwell-Boltzmann distribution, and is shown in figure 1 for argon gas at $T = 273$ K.

Figure 1. Maxwell-Boltzmann distribution of particle velocities for argon at $T=273$ K. The vertical line marks the average speed, $\langle v \rangle$.

2.4 Mean Free Path and Collision Frequency

While the velocity probability distribution for an equilibrium dilute gas system has been derived, expressions for several other quantities must be obtained before the DSMC method can be used. Two of these are the mean free paths of particles in the dilute gas, and the frequency of collisions.

The mean free path, λ , is the average distance through which a particle travels before colliding with another particle. It is related to the collision frequency, ν , as

$$\lambda = \langle v \rangle / \nu \quad (\text{EQ 12})$$

where $\langle v \rangle$ is the average particle speed.

The collision frequency is the average number of collisions per particle per unit time. To determine the collision frequency, the volume a particle sweeps out in a particular time step is determined, multiplied by the number of particles per unit volume, then normalized with respect to that time step. For a particle with a diameter, d_{eff} , the volume swept out during a time step is visualized in figure 2.

Figure 2. Particle moving amid a field of stationary particles.

The volume swept out is approximated by $\pi d_{\text{eff}}^2 l$, even though the cylinder has 'elbows' at the locations of collisions. If the particle has a velocity of v_r and the dilute gas is homogenous, then the number of stationary particles encountered by the moving particle is

$$\begin{aligned}
 M_{\text{tube}} &= (\text{Number of particles per unit volume}) \\
 &\quad \times (\text{Volume of cylinder}) \\
 &= (N/V) (\pi d_{\text{eff}}^2 v_r t_0)
 \end{aligned}
 \tag{EQ 13}$$

which is also the number of collisions in a time step, t_0 .

The collision frequency is the time average of M_{tube} and is given by

$$\nu = \frac{\langle M_{\text{tube}} \rangle}{t_0} = \frac{N}{V} \pi d_{\text{eff}}^2 \langle v_r \rangle
 \tag{EQ 14}$$

However, since all particles are moving, $\langle v_r \rangle$ should be identified as the average relative velocity between particles [9]. The mean free path is then

$$\lambda = \frac{\langle v \rangle}{v} = \frac{V}{N\pi d_{\text{eff}}^2} \frac{\langle v \rangle}{\langle v_r \rangle} \quad (\text{EQ 15})$$

2.5 Flow Parameters

There are several basic parameters that provide a convenient measure of certain flow or system values. Of these, the Knudsen number, viscosity, and Mach number are the most appropriate to define for our treatment of modeling dilute gas flows.

2.5.1 Knudsen Number

The Knudsen number is the relationship between the characteristic dimension of a system, L , to the mean free path of particles composing the system. The relationship is expressed as

$$\text{Kn} = \frac{\lambda}{L} \quad (\text{EQ 16})$$

This dimensionless number is often used as a criterion for determining when to use continuum fluid models or particle methods. The continuum region is denoted by the relationship between the number density, $n \propto 1/\lambda$, and the size of the system, L . Hydrodynamic equations, like Navier-Stokes, are only valid in the continuum region. The Knudsen number for a continuum fluid system is less than unity, but for most hydrodynamic systems, $(\text{Kn}) \leq 0.1$ is used as a more practical limit.

In figure 3, the continuum and particle methods regimes are shown for air under standard conditions. Both density and characteristic dimension are used to establish limits for the two regimes, with $(Kn) \leq 0.1$ used for the continuum region, and δ/d for a dilute gas [10].

Figure 3. The delineation of regimes for air at standard pressure and temperature as a function of density and characteristic length.

2.5.2 Viscosity

Viscosity is one of the most important parameters used to describe a fluid. The dynamic viscosity, η , is defined as the coefficient of proportionality between the momentum flux and the velocity gradient in the fluid,

$$\text{momentum flux} = -\eta \nabla u \quad (\text{EQ 17})$$

where u is the stream velocity. The kinematic viscosity, ν , is defined as the dynamic viscosity over the density, $\nu = \eta/\rho$. In section 3, viscosity is cal-

culated in the Couette flow system by the measurement of the macroscopic stream velocities.

2.5.3 Mach Number

The speed, a , by which transverse waves (e.g., sound) move through a fluid is another important parameter. The flow's Mach number, Ma , is the ratio of the stream flow, u , and a given by

$$Ma = \frac{|u|}{a} \quad (\text{EQ 18})$$

For an ideal monatomic gas, a is expressed as

$$a = \sqrt{\gamma \frac{kT}{m}} \quad (\text{EQ 19})$$

where $\gamma = c_p/c_v = 5/3$, k is the Boltzmann constant, T is the temperature, and m is the mass of one atom.

The Mach number is useful in characterizing the relative densities and momentum transport of fluids. For low Mach numbers ($Ma < 0.3$) the flow is incompressible, to a good approximation. At higher Mach numbers ($Ma > 1.0$), discontinuous effects like shock waves appear.

2.6 Boundary Conditions

In moving from theoretical models to modeling physically realizable systems, properly specifying the boundary interactions are required. While particle-particle interactions described in the previous section are important, a system's dynamics are often driven by boundary interactions.

While the following discussion presents a simple mechanical picture as a guide in describing boundary interactions, a more complete treatment involves quantum mechanics. When a particle strikes a boundary, such as a

wall, two interactions result from the surface imperfections of the wall. For example, when a particle strikes a 'peak' on a wall, as illustrated in figure 4, the interaction is modeled as an exchange of momentum between the wall and particle. Since the wall has a mass much larger than the particle, the collision of wall and particle is purely elastic and is called a specular collision. The particle merely reflects off the wall, having the velocity component perpendicular to the wall changed in sign.

The other, more frequent interaction, is a thermal energy transfer when the particle impacts the wall in a 'valley' or on the side of a peak. The particle remains in contact with the wall long enough to exchange energy and become thermalized by the wall, since the wall has some amount of internal energy. This is called a thermal collision. The particle reflects off the wall in a random direction, with a velocity that is a function of the temperature of the wall.

Figure 4. Boundary surface imperfections result in two types of collisions - specular and thermal.

In treating specular boundary collisions in a computational model, the particle's velocity component perpendicular to the boundary is changed in sign, while the components parallel to the boundary remain unchanged. For thermal boundary collisions, the boundary is held at a constant temperature, and is unaffected by the change in energy occurring when a particle strikes it. It is thus treated computationally as a thermal reservoir.

Consequently, describing a particle's interaction with a thermal boundary is more complex than for a specular boundary. The process of thermalization is used to compute the new kinetic energy of the particle. Since temperature itself is a macroscopic property, thermalization is usually approached as a statistical phenomenon. This means that the energy gained (or lost) is a function of temperature, but is applied to the kinetic energy of the particle in a stochastic manner. Therefore, the particle will leave the boundary in a random direction, as well as with a random speed [9].

The velocity distribution of particles reflecting off a fluxing boundary is given by

$$\begin{aligned}
 P(u, v, w) &= \frac{1}{C} w e^{-\beta^2((u-u_0)^2 + (v-v_0)^2 + (w-w_0)^2)} \\
 &= \left(\frac{1}{C_x} e^{-\beta^2(u-u_0)^2} \right) \left(\frac{1}{C_y} e^{-\beta^2(v-v_0)^2} \right) \left(\frac{1}{C_z} w e^{-\beta^2(w-w_0)^2} \right) \quad (\text{EQ 20}) \\
 &= P_x(u) P_y(v) P_z(w)
 \end{aligned}$$

where $w \in [0, \infty]$, $u, v \in [-\infty, \infty]$, and u_0, v_0 , and w_0 are wall velocities. The factor $\beta = \sqrt{m/(2kT_0)}$, m is the mass of the particles and the normalizations are

$$C_x = C_y = \frac{\sqrt{\pi}}{\beta}$$

$$C_z = \left[\frac{1}{2\beta^2} e^{-\beta^2 w_0^2} + \frac{\sqrt{\pi}}{2\beta} w_0 (1 + \operatorname{erf}(\beta w_0)) \right] \quad (\text{EQ 21})$$

Since the distribution can be decomposed as a product of the distributions for the three velocity components, each one can be computed separately. The x - and y -components are simple Gaussian distributions. The z -component, shown in figure 5 for $w_0 = 0$, is a biased Maxwell-Boltzmann distribution.

Figure 5. Velocity (z-component) probability distribution for particles leaving a thermal reservoir with $w_0 = 0$. Similar to Maxwell-Boltzmann distribution, except for different normalization constant, and w instead of w^2 factor. The vertical line represents the average velocity, $\langle w \rangle$.

2.7 Simple Dilute Gas Systems

This section presents three simple systems, one in equilibrium, and the other two out of equilibrium with constant velocity and temperature gradients. The computational model in section 3 describes both of these systems very well, showing the model's applicability to both equilibrium and non-equilibrium systems.

2.7.1 Thermodynamic Equilibrium

In thermodynamic equilibrium, no forces, wall velocities, or temperature gradients are present. The gas molecules only interact with each other and the system boundaries. Therefore, the flow velocities in each direction fluctuate about zero. While this flow is not particularly interesting, it serves as a good test case of fluid models and gives an indication of the time required for a system to reach equilibrium, useful in determining when a nonequilibrium system will reach a steady-state.

2.7.2 Couette Flow

A dilute gas contained between two walls moving in opposite directions with velocities $\pm u_w$ in the y -direction (see figure 6) is called planar Couette flow. The walls can be infinite planes, and the boundaries perpendicular to the wall can be periodic.

Figure 6. Planar Couette flow. Left and right walls move at constant velocities, but in opposite directions.

Even though this is a nonequilibrium system, the flow reaches a steady-state and can be described by the theory presented thus far. This application of particle equilibrium theory to nonequilibrium systems is the basis for the work presented in this paper. In the section describing the computer simulations, the value of viscosity obtained from modeling Couette flow using the DSMC method agrees very well with the more rigorous approach to nonequilibrium systems of the Chapman-Enskog theory.

From equation 17, the assumption that the net flux of y -momentum varies linearly with the velocity gradient introduced by the moving walls gives,

$$F_{p_y} = -\eta \frac{du_y}{dz} \quad (\text{EQ 22})$$

An approximate expression for the viscosity, η , of this system can be obtained using a heuristic argument first proposed by Maxwell. He considered a vertical plane located at $z = z^*$, as shown in figure 7. From purely dimensional arguments, the total flux of particles crossing this plane from right to left is $\alpha \langle v \rangle n$ where n is the number density and α is a constant of order unity.

Figure 7. Schematic for Maxwell's estimate of viscosity.

Particles crossing the boundary had their last contact with another particle about one mean free path from the plane. Since the y -velocity of the fluid is $u_y(z)$, those particles on the right, on average, carry a y -momentum of

$mu_y(z^* + \lambda)$, while those crossing from left to right, carry a y -momentum of $mu_y(z^* - \lambda)$. So, the y -momentum flux is approximately

$$\begin{aligned}
 F_{p_y} &= [\alpha \langle v \rangle n m u_y(z^* - \lambda)] - [\alpha \langle v \rangle n m u_y(z^* + \lambda)] \\
 &\approx -2\alpha \langle v \rangle n m \lambda \frac{u_y(z^* + \lambda) - u_y(z^* - \lambda)}{2\lambda} \\
 &\approx -2\alpha \langle v \rangle n m \lambda \left. \frac{du_y}{dz} \right|_{z=z^*}
 \end{aligned} \tag{EQ 23}$$

From equation 22, the viscosity is

$$\eta \approx 2\alpha m n \langle v \rangle \lambda \tag{EQ 24}$$

where, after more sophisticated derivations, α is found to lie between $1/4$ and $1/6$ [9]. The viscosity derived in the Chapman-Enskog theory [11] is

$$\eta = \frac{5\pi}{32} m n \langle v \rangle \lambda \tag{EQ 25}$$

2.7.3 Temperature Gradient

By fixing the walls of a system at different temperatures, a temperature gradient can be modeled in a dilute gas. As the system progresses to a steady-state, this temperature gradient causes a redistribution of gas molecules so that pressure ($p \propto nT$, where n represents the number of molecules in a cell) is constant across the system.

While the y -velocity distribution for Couette flow is linear, the thermal diffusivity, κ , varies with temperature as

$$\kappa(T) = \kappa_0 \sqrt{T} \tag{EQ 26}$$

where κ_0 is constant, for a gas where molecules are modeled as hard spheres. Using this expression for κ and solving the diffusion equation for a steady-state condition

$$\frac{\partial T}{\partial t} = \frac{\partial}{\partial z} \kappa \frac{\partial T}{\partial z} = 0 \quad (\text{EQ 27})$$

gives

$$T^{3/2} = (T_1^{3/2} - T_0^{3/2}) \frac{z}{L} + T_0^{3/2} \quad (\text{EQ 28})$$

where $T_0 = T(z = 0)$ and $T_1 = T(z = L)$ are the wall temperatures, L is the wall separation distance, and z is some point between the walls. These results for the temperature profile are presented in figure 8, along with a linearly varying temperature gradient for comparison.

Figure 8. Comparison of temperature profiles for hard sphere thermal diffusivity (solid line) and constant thermal diffusivity (dashed line).

CHAPTER 3

COMPUTATIONAL METHODS

There are, in general, two approaches in establishing computational models that describe fluid motion. The first involves treating the fluid as a continuum, i.e., neglecting the molecules that comprise the fluid, using the conservation laws. The second approach is to take into account the statistical behavior of individual fluid molecules using the kinetic theory of gases. Each of these approaches is an approximation of the actual fluid, having different limitations and areas of applicability.

3.1 Continuum-Based Analytic Equations

Treating the fluid as a continuous medium results in deriving equations based on defining a definite fixed region or volume in space, as opposed to identifying and following fixed quantities of matter that make up the fluid. However, the three basic laws of conservation of mass, momentum, and energy do not apply directly to fixed regions or volumes, but to fixed quantities of matter. The typical derivation process is to develop the integral equations for a control volume. The integral equations are then converted to differential equations using vector calculus, or by taking the limit of the control volume as it approaches zero. This process relates physical properties of the fluid, not the control volume, to the fluid's motion.

3.1.1 Conservation of Mass

Following this methodology, applying the conservation of mass to a control volume yields the continuity equation

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{V}) = 0 \quad (\text{EQ 29})$$

where ρ is the fluid density and $\mathbf{V}$ is the fluid velocity. This equation is often expressed in the form

$$\frac{D\rho}{Dt} + \rho (\nabla \cdot \mathbf{V}) = 0 \quad (\text{EQ 30})$$

where the *substantial* derivative

$$\frac{D()}{Dt} \equiv \frac{\partial}{\partial t}() + \mathbf{V} \cdot \nabla () \quad (\text{EQ 31})$$

is used [6].

3.1.2 Conservation of Momentum

Conservation of momentum is used to derive the equation

$$\frac{\partial}{\partial t}(\rho \mathbf{V}) + \nabla \cdot \rho \mathbf{V} \mathbf{V} = \rho \mathbf{f} + \nabla \cdot \Pi_{ij} \quad (\text{EQ 32})$$

where $\rho \mathbf{f}$ is body force per unit volume. The most common body force acting on a fluid is gravity. The surface forces per unit volume are represented by $\nabla \cdot \Pi_{ij}$. Approximations to the stress tensor, Π_{ij} , for all gases that can be treated as a continuum, and most liquids, yields the *Navier-Stokes equation*

$$\rho \frac{D\mathbf{V}}{Dt} = \rho \mathbf{f} - \nabla p + \frac{\partial}{\partial x_j} \left[\mu \left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} \right) - \frac{2}{3} \delta_{ij} \mu \frac{\partial u_k}{\partial x_k} \right] \quad (\text{EQ 33})$$

which is also called the *viscous momentum equation*.

In order to show the development of the equations that relate a fluid's physical properties to its motion from integral equations that are based on a control volume, the Navier-Stokes equation is now derived [12].

The momentum equation is merely Newton's second law where the net force $\mathbf{F}$ acting on a particle or system of particles of fixed mass is given by

$$\mathbf{F} = \frac{d}{dt} \mathbf{M} \quad (\text{EQ 34})$$

where $\mathbf{M}$ is the total linear momentum of the particle or system of particles.

Applying equation 34 to the control volume in figure 9, and assuming F is constant over a time interval, Δt , gives

$$F \Delta t = \Delta M = \frac{M_A(t + \Delta t) - M_A(t)}{\Delta t} + \frac{M_B(t + \Delta t) - M_{AC}(t + \Delta t)}{\Delta t} \quad (\text{EQ 35})$$

Figure 9. Fluid moving through a control volume.

Re-arranging equation 35, dividing by Δt , and taking the limit as $\Delta t \rightarrow 0$ gives

$$F = \frac{\partial}{\partial t} (M)_{CV} + \sum_B \Delta \dot{M} - \sum_C \Delta \dot{M} \quad (\text{EQ 36})$$

where CV is the control volume, the first term on the right is the rate of change of momentum within the control volume, and the second and third terms are the momenta associated with mass that is flowing out of and into the control volume, respectively. Since M is the total linear momentum in the control volume, the first term can be expressed as

$$\frac{\partial}{\partial t} (M)_{CV} = \frac{\partial}{\partial t} \int_{CV} \rho \mathbf{V} dV \quad (\text{EQ 37})$$

and the second and third terms become

$$\sum_B \Delta \dot{M} - \sum_C \Delta \dot{M} = \left[\sum \Delta \dot{m} \mathbf{V} \right]_{out} - \left[\sum \Delta \dot{m} \mathbf{V} \right]_{in} \quad (\text{EQ 38})$$

From figure 9, $\dot{m}$ represents the mass rate of flow in or out of the control volume with a velocity, $\mathbf{V}$. This gives the following expression for $\mathbf{F}$:

$$\mathbf{F}_s + \int_{CV} \mathbf{B} dV = \frac{\partial}{\partial t} \int_{CV} \mathbf{V} \rho dV + \int_{CS} \mathbf{V} \rho \mathbf{V} \cdot d\mathbf{A} \quad (\text{EQ 39})$$

where $\mathbf{F}_s$ is the total surface force, composed of pressure and shear forces, and $\mathbf{B}$ is the body force (pf) per unit volume.

By applying this integral equation to an elemental volume, the Navier-Stokes equation can be derived. First, consider figure 10, where a cube represents an elemental volume. The stresses shown in figure 10 are in the x -direction, which are components of the overall stress tensor

$$\sigma_{ij} = \begin{bmatrix} \sigma_{11} & \sigma_{12} & \sigma_{13} \\ \sigma_{21} & \sigma_{22} & \sigma_{23} \\ \sigma_{31} & \sigma_{32} & \sigma_{33} \end{bmatrix} \quad (\text{EQ 40})$$

The momentum balance in the x -direction is then

$$\begin{aligned} & \left(\sigma_{11}|_{x+\Delta x} - \sigma_{11}|_x \right) \Delta y \Delta z + \left(\sigma_{21}|_{y+\Delta y} - \sigma_{21}|_y \right) \Delta x \Delta z + \left(\sigma_{31}|_{z+\Delta z} - \sigma_{31}|_z \right) \Delta x \Delta y \\ & + B_x \Delta x \Delta y \Delta z = \frac{\partial}{\partial t} (u \rho) \Delta x \Delta y \Delta z + \Delta y \Delta z (\rho u^2|_{x+\Delta x} - \rho u^2|_x) \\ & + \Delta x \Delta z (\rho uv|_{y+\Delta y} - \rho uv|_y) + \Delta x \Delta y (\rho uw|_{z+\Delta z} - \rho uw|_z) \end{aligned} \quad (\text{EQ 41})$$

Dividing through by $\Delta x \Delta y \Delta z$ and taking the limit as Δx , Δy , and $\Delta z \rightarrow 0$, and combining with the continuity equation, equation 29, the following is obtained for the Navier-Stokes equation in the x -direction:

$$\rho \frac{Du}{Dt} = \rho \left(\frac{\partial u}{\partial t} + u \frac{\partial u}{\partial x} + v \frac{\partial u}{\partial y} + w \frac{\partial u}{\partial z} \right) = \frac{\partial \sigma_{11}}{\partial x} + \frac{\partial \sigma_{12}}{\partial y} + \frac{\partial \sigma_{13}}{\partial z} + B_x \quad (\text{EQ 42})$$

Figure 10. Elemental volume for momentum equation. Only stress forces in the x -direction are shown. The y - and z -components are obtained in a similar manner.

3.1.3 Conservation of Energy

By applying the conservation of energy to a control volume, and then transforming to differential form, gives

$$\frac{\partial E_t}{\partial t} + \nabla \cdot E_t \mathbf{V} = \frac{\partial Q}{\partial t} - \nabla \cdot \mathbf{q} + \rho \mathbf{f} \cdot \mathbf{V} + \nabla \cdot (\Pi_{ij} \cdot \mathbf{V}) \quad (\text{EQ 43})$$

where E_t is the total energy per unit volume, q is the heat transfer, and is equal $-k \nabla T$, where k is the coefficient of thermal conductivity and T is the

temperature. The derivation of the conservation of energy is similar to the derivation of the viscous momentum equation and is not presented here.

These three equations, the continuity equation, the viscous momentum equation, and the energy equation are commonly known as the *Navier-Stokes equations*. However, sometimes only the components of the viscous momentum equation are called the Navier-Stokes equations.

3.1.4 Solving the Equations

Several methods are employed to solve these equations computationally. Two of the most popular are variants of the relaxation method, used to solve elliptic partial differential equations, and marching methods, used to solve hyperbolic or parabolic partial differential equations. These methods approximate the derivatives in the Navier Stokes equations by using finite differences, where derivatives are replaced by functions that approach the derivatives in the limit. For example,

$$\frac{d\rho}{dx} \approx \frac{\rho(x + \Delta x) - \rho(x)}{\Delta x} \quad (\text{EQ 44})$$

where Δx is the step size of a grid of points, or an interval in time.

Relaxation methods have several limitations. The chief of these limitations is the requirement for time-independence (steady-state condition) of the fluid equations. In order to speed convergence, artificial time dependence is usually introduced to embed the steady-state equation in a convergent time-dependent process [13]. Another short-coming is their inability to model rotational flow.

Marching methods, unlike relaxation methods, allow for a slow progression (or march) in time to be modeled. This means that marching methods

return a complete description of the fluid flow after every time step, in contrast to relaxation methods, which only return a valid solution once they converge.

A marching method works by converting a simple, coupled set of hydrodynamic partial differential equations (i.e., Navier-Stokes) given by

$$\frac{\partial S}{\partial t} = G(S(\mathbf{r}, t)) \quad (\text{EQ 45})$$

where S is a function of position, $\mathbf{r}$, and time, t , into finite difference form

$$\frac{S(\mathbf{r}, t + \Delta t) - S(\mathbf{r}, t)}{\Delta t} = G(S(\mathbf{r}, t)) \quad (\text{EQ 46})$$

$$S(\mathbf{r}, t + \Delta t) = S(\mathbf{r}, t) + \Delta t G(S(\mathbf{r}, t))$$

where the left-hand side represents the 'new' state property, like temperature or density, and the right hand side represents the sum of the old state property plus a function, G , relating other dependent parameters including past states.

Numerical implementations of relaxation and marching methods are very dependent on boundary conditions, requiring an algorithm to be redesigned for changing boundary conditions in order to maintain convergence. Also, the Navier-Stokes equations are only valid when the physical system can be treated as a continuum. The continuum approximation stipulates that the individual particle interactions on system dynamics are negligible. Statistically, this means the continuum approach is only valid if there are sufficient particles in a volume element to compute useful averages of the flow's properties. The limit at which the continuum approximation breaks down is for a Knudsen number of order unity. At that point, sufficient variations between the predicted outcome and that for a real

fluid begin. However, even with these limitations, these methods have been quite successful at modeling transonic, small-disturbance flows [14].

3.2 Direct Particle Simulations

Both Molecular Dynamics and DSMC are direct particle simulation methods derived from kinetic gas theory. The algorithms they employ provide a direct physical simulation of a fluid flow, rather than a numerical solution of equations as in done in Navier-Stokes methods. The primary difference between the two direct methods is that Molecular Dynamics (MD) is deterministic, i.e., all particles in the fluid are modeled, while DSMC is probabilistic in that each simulation particle represents an *ensemble* of particles.

3.2.1 Molecular Dynamics

The MD method mentioned in section 2 was the one of the first direct particle simulation methods. Developed by Alder and Wainwright in 1957 [15], it models the qualitative properties of a dynamic fluid system through the time-incremental computation of individual particle trajectories and interactions.

This method differs from the relaxation and marching methods for solving the hydrodynamic equations by using simple Newtonian formulations (equation 2). By mapping a set of 'useful' physical characteristics and boundary conditions onto a simulation space, MD simulates the fluid flow within that space. The algorithm moves the particles using Newtonian dynamics, tracks each particle as it is moved, and determines the interactions with other particles and the boundaries.

Essentially, the MD algorithm is composed of the following steps:

- Partition the simulation space into equally sized cells or evenly spaced lattice points.
- Distribute particles within the simulation space, giving each particle some initial velocity from a distribution of velocities.
- Determine particle-particle interactions by checking the proximity of each particle to every other particle in the simulation. For all particle pairs that interact, calculate their interaction force using a particle-potential function, such as a Lennard-Jones potential, and determine the corresponding trajectories.
- Move all particles in the simulation one time step using Newtonian dynamics, storing the position and velocity values. Since the Newtonian equations can be expressed as simple ODE's, a Runge-Kutta-type algorithm is efficient to use.
- Recalculate the positions and velocities of particles crossing a boundary.
- Gather statistical information to determine macroscopic fluid properties at end of the simulation.

Determining the proper time step to use in moving the simulation particles is particularly important in particle simulation models. In the MD algorithm, it is possible to violate energy conservation if the time step is not small enough to handle the collisions of the faster particles in the system. Such particles could find themselves too close to other particles, thereby gaining an unrealistic amount of kinetic energy when a particle-potential function like the Lennard-Jones potential is used, which increases rapidly for small separation distances [16].

3.2.2 Direct Simulation Monte Carlo

The DSMC method, originally proposed by Bird in 1963 [17], employs simulation particles as the MD method does, but the number of particles in the system is reduced to a manageable size by regarding each simulated particle as representing an arbitrarily fixed number of real molecules. The collision frequency parameter uses this attribute of the DSMC method.

As with the MD method, the DSMC algorithm divides the simulation space into equally sized cells, and particle movement is uncoupled from interparticle collisions (potential interactions for the MD algorithm) over the time step. The procedure for selecting which particles collide is based directly on the relations that have formed the basis of kinetic theory for more than a century [18]. In fact, the assumptions of molecular chaos and the requirement of a dilute gas are the same assumptions the Boltzmann equation is based on, and the DSMC algorithm is equivalent to a numerical solution of this equation [19]. The assumption of a dilute gas takes the form of requiring a mean free path large in comparison with the mean spacing between molecules, which is large compared to the molecular diameter. This has the direct effect that collisions occur only between two particles.

The following steps summarize the DSMC algorithm:

- Divide the simulation space into equally sized cells, and evenly distribute the particles, with a given velocity distribution.
- Move the particles using a very simple routine, and process particle-boundary interactions.
- Determine the number of collisions possible, then select pairs meeting a certain randomly chosen criterion for collision.
- Determine post-collision velocities for each collision pair, *but not positions*.
- Sample particle properties, such as position, velocity, and temperature, for determining macroscopic fluid properties at end of run.

An essential feature of Bird's DSMC method is the method in which particles are chosen for collision. Cell sizes are fixed to be less than one mean free path, so that exact particle locations within the cell are not required. Pairs of particles are selected randomly, and the likelihood of collision is based on the relative velocity between the pair. This is the basic difference between the MD and DSMC algorithms: that particle positions in MD must be tracked to determine every interaction, while DSMC associates the

probability of interaction with relative velocity when particles are within the same computational cell [20].

3.3 Comparison of Methods

The MD algorithm is far more general in its applicability to modeling all fluids, regardless of density. The DSMC algorithm, however, only applies to dilute gases satisfying the Boltzmann equation, which necessitates binary collisions.

So, while the region where DSMC methods apply is limited, within that region it is the particle method of choice. For a system with the same number of particles, smaller computational resources than the MD algorithm are required since each particle represents a large number of similar molecules, and exact particle positions are not required in processing collisions, thus reducing memory requirements and computer run time [21], [22].

CHAPTER 4

DSMC SIMULATIONS

4.1 Code Description

The direct particle simulation model described here is based on an algorithm developed by G. A. Bird. In this model, a large number of particles are modeled by a significantly smaller number of representative particles of the same size. An adaptation of the algorithm [9] in the MATLAB programming language was translated into the C language and augmented with the addition of buffer cells and attendant statistics gathering to describe boundary crossing velocity profiles. The implementation of the buffer cells is such that a constant density can be maintained from time step to time step (but allowed to vary during a single time step) in order to simulate the presence of analytic-type cells typical of a Navier-Stokes model, where aggregate properties of density and average velocity are used. This patching scheme between the two methods is important in describing a dilute gas flowing from a region where analytic models are valid (continuum regime) to regions where particle simulations are required. The program also has an option, by setting the buffer cell densities to -1 , that allows the density to vary from time step to step, just like a regular simulation cell.

The code first reads in problem set-up information describing average number of particles per cell, the number of cells, the velocity of the walls, whether the walls act as thermal reservoirs or as specular walls, and the temperature of the walls (if the walls are thermal). Buffer cell information is next read in, setting those cells in the system (if any) that have fixed parameters, the velocity bias in those buffer cells, the number of particles in each buffer cell, and the temperature in the buffer cells. Next, the name

of the file containing particle positions and velocities from a previous set of runs are read in, along with a flag as to whether or not to use the restart file, the number of runs, the number of time steps per run, the initial random number seed, and the file that will be written with the final particle positions and velocities to provide a restart capability.

For a new set of runs, particle positions are randomly generated throughout the system. Each particle is given a velocity taken from a Gaussian distribution, and an initial velocity or temperature gradient profile if the walls have non-zero velocities or temperatures, respectively. The number of particles in the buffer cells are then reset to the appropriate density read from the set-up file, unless -1 was input for the number of particles within a particular buffer cell. Each of the particles is again given a velocity selected randomly from a Gaussian distribution, modified by the temperature and velocity in that particular buffer cell. The time step, t_0 , is selected to be 20% of a cell length travelled by a particle moving with the most probable velocity, v_{mp} ,

$$t_0 = \frac{0.2l}{v_{mp}} \quad (\text{EQ 47})$$

where l is the system length, L , divided by the number of cells. The most probable velocity, v_{mp} , is the velocity at which the Maxwell-Boltzmann probability distribution has its maximum,

$$\left. \frac{d}{dv} p(v) \right|_{v=v_{mp}} = 0 \Rightarrow v_{mp} = \sqrt{2} \sqrt{\frac{kT}{m}} \quad (\text{EQ 48})$$

The most probable velocity is also determined for each of the buffer cells, based on the temperature read from the start-up file. During the course of a

run, the velocities of particles in buffer cells are reset from a Gaussian distribution, multiplied by the v_{mp} for that particular buffer cell.

All particles are then moved, and checked to see if any have crossed into or out of a buffer cell. If any have, the z -component of velocity is sampled and stored in a histogram to determine the distribution of particle velocities crossing the buffer cell boundaries. If a particle strikes either wall, the z -component of its velocity is thermalized for thermal reservoir walls, or changed in sign for specular walls. For walls modeled by thermal reservoirs, the x - and y -components of velocity are randomly chosen from a Gaussian distribution, and the y -component is modified by the wall velocity.

The next step is to determine the number of particle collisions for each cell. The cell dimension has been chosen to be on the order of one mean free path so that the exact positions of the particles within the cell are not required. The number of possible collisions for each cell is found by using the collision frequency with a modified determination of the probability of a collision occurring within the cell.

The collision frequency in a cell is dependent on the actual number of particles within a computational cell. So, for the purpose of determining the collision frequency for this model, each simulation particle has an effective cross-section of $F_n \pi d_{eff}^2$, giving a collision frequency of (using equation 14)

$$\nu = \frac{N_c}{V_c} F_n \pi d_{eff}^2 \langle v_r \rangle \quad (\text{EQ 49})$$

where F_n is the number of actual particles each simulation particle represents, N_c is the number of particles in a cell, and V_c is the volume of one cell.

The total number of collisions in a cell during the time step, t_0 , is

$$M_{coll} = \frac{1}{2} N_c v t_0 = \frac{N_c^2 F_n \pi d_{eff}^2 \langle v_r \rangle t_0}{2 V_c} \quad (\text{EQ 50})$$

The modified probability of collision is computed using the acceptance-rejection criterion

$$\frac{|\mathbf{v}_i - \mathbf{v}_j|}{v_r^{max}} > \mathfrak{R} \quad (\text{EQ 51})$$

where v_r^{max} is the maximum relative velocity in the cell, and $\mathfrak{R}$ is a uniform random deviate. If the pair is accepted, the collision is processed, otherwise another pair of particles is randomly chosen and their relative velocity screened by equation 51. The accept-reject criterion of the above equation replaces the more computationally intensive discrete probability distribution

$$P_{coll}(i, j) = \frac{|\mathbf{v}_i - \mathbf{v}_j|}{N_c \sum_{m=1}^{m-1} \sum_{n=1} |\mathbf{v}_m - \mathbf{v}_n|} \quad (\text{EQ 52})$$

where the probability is normalized by the double sum in the denominator.

Using equation 51 as the probability of collision is the basis of the NTC (no time counter) collision sampling method developed by Bird [18]. He found that the NTC method, with the probability of collision given by equation

51, gave the exact collision rate in both simple gases and gas mixtures, and in thermodynamic equilibrium or nonequilibrium conditions.

Finally, the number of possible collisions that occur at each time step is obtained from setting the ratio of total accepted candidates to total possible candidates equal to the probability of accepting a pair for collision, which is related to their relative velocity,

$$\frac{M_{\text{col}}}{M_{\text{cand}}} = \frac{\langle v_r \rangle}{v_r^{\text{max}}} \quad (\text{EQ 53})$$

Using equation 50 and equation 53 gives

$$M_{\text{cand}} = \frac{N_c F_n \pi d_{\text{eff}}^2 v_r^{\text{max}} t_0}{2V_c} \quad (\text{EQ 54})$$

which is the number of candidate pairs of particles selected for collision during each time step.

The initial selection of v_r^{max} can be somewhat arbitrary, since its actual value does not affect the collision rate, as it appears in the denominator of equation 51 and the numerator of equation 54. If set too large, however, processing collisions will become inefficient, since there will be a large number of candidate collisions, M_{cand} , with few of those candidates meeting the collision criterion $|v_i - v_j|/v_r^{\text{max}}$. In this simulation, v_r^{max} is chosen to be $3v_{\text{mp}}$ for each cell, with the provision of being updated should a greater value occur.

The post-collision velocities of a pair of particles passing the collision criterion are evaluated by considering conservation of linear momentum. The

center of mass velocity remains unchanged, as does the relative velocity magnitude,

$$\mathbf{v}_{cm} = \frac{1}{2}(\mathbf{v}_i + \mathbf{v}_j) = \frac{1}{2}(\mathbf{v}_i^* + \mathbf{v}_j^*) \quad (\text{EQ 55})$$

where the * represents post-collision values.

From the conservation of energy, the magnitude of the relative velocity also remains unchanged and is given by

$$v_r = |\mathbf{v}_i - \mathbf{v}_j| = |\mathbf{v}_i^* - \mathbf{v}_j^*| = v_r^* \quad (\text{EQ 56})$$

The remaining unknowns are fixed by the angles, θ and ϕ , for the relative velocity,

$$\mathbf{v}_r^* = v_r [(\sin\theta \cos\phi)\mathbf{i} + (\sin\theta \sin\phi)\mathbf{j} + \cos\theta\mathbf{k}] \quad (\text{EQ 57})$$

where $\mathbf{i}$, $\mathbf{j}$, and $\mathbf{k}$ are the unit vectors in the x -, y -, and z -directions, respectively. The angles θ , and ϕ , for the hard sphere model, are uniformly distributed over the unit sphere. The azimuthal angle, ϕ , is uniformly distributed between 0 and 2π , and θ is distributed according to the probability density,

$$\begin{aligned} \phi &= 2\pi\mathfrak{R} \\ P(\theta)d\theta &= \cos\theta d\theta \end{aligned} \quad (\text{EQ 58})$$

Using the change of variable, $q = \sin\theta$, gives $P(q)dq = 1dq$, so that q is uniformly distributed in the interval $[-1,1]$. Setting $q = 2\mathfrak{R} - 1$ gives

$$\begin{aligned} \sin\theta &= q \\ \cos\theta &= \sqrt{1 - q^2} \end{aligned} \quad (\text{EQ 59})$$

so that a direct calculation of θ is not required in evaluating equation 57.

The post-collision velocities are then

$$\begin{aligned} \mathbf{v}_i^* &= \mathbf{v}_{cm}^* + \frac{1}{2}\mathbf{v}_r^* \\ \mathbf{v}_j^* &= \mathbf{v}_{cm}^* - \frac{1}{2}\mathbf{v}_r^* \end{aligned} \tag{EQ 60}$$

Once all collisions within each cell have been processed, sampled information is gathered. The fluid properties measured are the average number of particles within a cell, average v_x , v_y and v_z velocities, and the average cell temperature.

Post-processing of the sampled information is performed in a separate program that determines the average quantities for the series of runs, and the standard deviations used in calculating error bars. For the buffer cells, velocity information is gathered for those particles crossing its boundaries by tallying v_z in bins from $-1000m/s$ to $1000m/s$. Particle velocities outside this region are also counted, but only to make sure that all particles entering or exiting the system are accounted for. The normalized distributions are calculated for comparison with the biased Maxwell-Boltzmann distribution of figure 5.

4.2 Thermodynamic Equilibrium Simulation

For simulation of thermodynamic equilibrium, the particles are evenly distributed within the simulation volume, and given an initial Maxwellian velocity distribution in all directions. The particles are moved, collided with each other and statistics are gathered on the number of particles within a cell, average velocity in the three dimensions for each cell, and average temperature within each cell.

4.3 Couette Flow Simulation

For Couette flow, the particles are evenly distributed within the simulation volume; however, the v_y for each particle is linearly biased by the wall velocity, helping to speed convergence of the system to steady-state. The wall collisions are necessarily thermal; therefore, the v_y for particles striking a wall are adjusted by the wall's velocity after being thermally randomized.

The viscosity is measured from the change in momentum of particles that reflect off the walls. From momentum-impulse theory, this flux is related to the time-average drag-force on a wall as

$$\langle f_{\text{drag}} \rangle = -\frac{1}{t} \sum m \Delta v_y = -\langle F_{p_y} \rangle \quad (\text{EQ 61})$$

where Δv_y is the change in y -velocity for a particle striking a wall. The average viscosity is then

$$\eta = \frac{\langle f_{\text{drag}} \rangle}{du_y/dx} \quad (\text{EQ 62})$$

The viscosity reported by the program is the average of the viscosities at each wall [9].

4.4 Temperature Gradient Simulation

For this simulation, the particles are also evenly distributed within the simulation volume, and the velocities for each particle are randomly chosen from a Gaussian distribution modified by the most probable velocity, v_{mp} . The v_{mp} for each cell is initially set according to a linearly varying temperature across the simulation volume. This puts the system near its steady-state condition, as determined from equation 28.

In order to try to match results between simulations with and without buffer cells, wall positions of the system without buffer cells were chosen to align with the midpoint of the first and last cells of the buffer cell system. This in effect added one cell to the buffer cell system, increasing the length by $0.05\text{ }\mu\text{m}$. How the two systems “line-up” is shown in figure 11.

Figure 11. Alignment of buffer cell and non-buffer cell systems.

CHAPTER 5

SIMULATION RESULTS

There were five separate simulation cases with and without buffer cells: equilibrium; Couette flow (constant velocity gradient); and a flow with a temperature gradient. Again, the presence of buffer cells is used to provide an alternate method for implementing boundary conditions. Buffer cells are also useful for linking an analytic method like Navier-Stokes, with the Direct Simulation Monte Carlo particle method.

In these simulations, 100 separate runs were made for each case, with each run consisting of 5,000 time steps. Samples were taken of various particle properties every time step after 500 time steps had been completed for each run. A separate data analysis program was coded to combine the 100 runs into one data set that contained an average and the error bars associated with the 100 runs for those quantities measured. The system length was set at 1 μm , with 20 cells (21 for buffer cell simulations), and a total of 1000 particles (1050 initial particles for fixed density buffer cell simulations).

The quantities measured were average number of particles in a cell, average x -, y -, and z -components of velocity (measured in two ways), average temperature in each cell, and the v_z distributions of particles crossing buffer cells boundaries.

In sampling the velocities to obtain average values over the course of a run, two methods were employed. The first method samples just velocities, which consists of summing the respective velocity components over all sampled time steps, and then dividing by the number of samples (4,500), yielding $\langle v \rangle$. The second method determines the thermodynamic velocity,

which is the sum of momentum within each cell divided by the time averaged mass in a cell,

$$\langle \mathbf{v} \rangle_{th} = \frac{\langle m \mathbf{v} \rangle}{\langle m \rangle} \quad (\text{EQ 63})$$

Comparison of these two quantities for the various simulations will be made to show that for certain simulations (those involving buffer cells where the density within the buffer cell is fixed), the thermodynamic time averaged velocity is the correct measure of average velocity.

Originally, the DSMC algorithm implemented in this work considered fixed temperature and density within the buffer cells, with the optional fixing of v_y . However, during the course of the temperature gradient simulations, it became evident that fixing the density had a large impact on the v_z results. The algorithm was subsequently modified to require only the setting of temperature in the buffer cells, leaving the setting of density and v_y as user options.

5.1 Thermodynamic Equilibrium Results

The results for thermodynamic equilibrium are presented in the next series of figures. The results reflect an initial uniform particle distribution, with particle velocities given a Maxwell-Boltzmann distribution about a system temperature of 273 K. The results are for 1,000 particles (50 per cell) and 500,000 time steps ($\sim 10\mu s$).

As mentioned earlier, the midpoint of the buffer cells corresponds to the walls in the non-buffer cell system. The introduction of buffer cells causes a slight increase in the number of particles per cell than the non-buffer cell results. However, the fluctuations about 50 are comparable for both cases.

Figure 12. Average density results of thermodynamic equilibrium case, with and without buffer cells.

The next figure, figure 13, shows the z-component of velocity results for both systems. These results indicate that a slight pressure gradient is present in the buffer cell system, although the velocity drift is small.

Figure 13. Average z-component of velocity across system.

What is interesting, however, is the difference in computing the average velocity using the thermodynamic equation (equation 63), the results of which are shown in figure 14.

Figure 14. Instantaneous average velocity and average velocity computed from instantaneous average momentum.

The “jumps” in average velocity from the buffer cells to non-buffer cells are considerably less, overall, than using instantaneous velocity to compute the average velocity. Also, the result for cell 21 (midpoint at 1.025 μm) shows the effect of the slight positive system velocity on the average v_z in that cell, increasing its value above the expected value of zero.

The results for the other velocity components are presented for completeness. However, only velocity in the z-direction has boundaries and hence any physical meaning.

For the v_y component, shown in figure 15, both systems exhibit “equilibrium-like” distributions. The buffer cell simulation results vary about zero velocity more so than the non-buffer cell simulation, but not unusually so.

Figure 15. Average y -component of velocity across system.

Similar results for the x -component are shown in figure 16. Again, the non-buffer cell simulation results are grouped slightly better about zero velocity. The average v_x in the buffer cells (cells 1 and 21) are nearly zero, indicative of the resetting of velocity components for particles in those cells.

Figure 16. Average x -component of velocity in each cell.

The average temperature across each system is presented in figure 17. The average temperatures in cells 1 and 20 for the non-buffer cell highlight the difference between thermalizing just those particles that interact with the wall and thermalizing all particles within a buffer cell.

Figure 17. Average temperature within each cell.

Velocity distribution data were also gathered for those particles crossing the buffer cell boundaries. The velocities were sampled and tallied in a histogram that spanned -1000m/s to 1000m/s in increments of 50m/s . These histograms were normalized with respect to number of particles with velocities less than or greater than zero, and plotted against the biased Maxwell-Boltzmann distribution of particles leaving a thermal reservoir (see figure 5). The distributions for those particles with positive and negative velocities crossing the cell 1 boundary are shown in figure 18 and figure 19. Distributions for particles crossing cell 21's boundary are shown in figure 20 and figure 21.

Figure 18. Probability density for particles crossing into cell 1.

Figure 19. Probability distribution for particles crossing out of cell 1.

Figure 20. Probability distribution for particles crossing out of cell 21.

Figure 21. Probability distribution for particles crossing into cell 21.

5.2 Couette Flow

While not an equilibrium system, Couette flow does reach a steady-state condition. The principal features of this flow are the linear velocity gradient in the y -direction, and the introduction of a shearing force because of the moving walls. In the density and temperature plots, viscous heating of the fluid is apparent in the middle of the system. There is a subsequent drop in average density in the center in order to maintain a constant pressure across the system.

Results are presented for systems with and without buffer cells. Viscosity obtained from the non-buffer simulation compared very well with the Chapman-Enskog theory for a hard sphere gas. The number of particles per cell as a function of system position is presented in figure 22. The num-

ber of particles in both simulations is lower in the middle than at the outside, showing an influx of heat into the system from the moving walls, which is more apparent in the temperature results, shown in figure 23. The results for the system with buffer cells are similar, yet show some important differences. For this system, the wall velocities were set to zero, and the wall was set as a specular boundary, not a thermal boundary. The buffer cells are the first and last, cells 1 and 21, respectively. The system length has also been increased by one cell length, aligning the midpoint of the buffer cells with the boundaries in the previous system that lacked buffer cells. The buffer cells densities have been fixed at 50 particles each (since the program as implemented only allows integral values for the particle density), with temperatures fixed at 273 K.

Figure 22. Overlay of particle densities for non-buffer and fixed density buffer cell Couette flows.

Since the number of particles for this system remain fixed, the decrease of particles in the middle of the system causes an increase in the average particle values for those cells near the walls. In the non-buffer cell simulation, the number of particles in the cells adjacent to the boundaries were free to fluctuate, and obtained a steady-state value of approximately 50.3 particles. This was the result of the heating caused by the moving walls propagating into the system. For the simulation with buffer cells using a fixed particle density, the same phenomenon occurs, causing particles to be removed from the system. The average number of particles in the non-buffer cell simulation was exactly 1000 (50 particles times 20 cells) for the system. For the fixed density buffer cell simulation, there were 1050 particles initially (50 times 21). After 500,000 time steps, the average was 1037.7 particles, a loss of 12 simulation particles removed by the buffer cells.

The temperature distribution for the buffer cell simulation shows the same behavior as the non-buffer cell simulation. An overlay of the two simulations respective temperature profiles are shown in figure 23.

Figure 23. Overlay of temperature distributions for non-buffer and fixed density buffer cell Couette flows.

The velocity results computed using the average instantaneous velocity at each time step are presented next. These results show nothing unexpected, as v_z across the system is essentially zero, the y -component of velocity shows the gradient introduced by the moving walls, and the x -component is shown for completeness; no boundaries affect its value. For v_z , the distributions are similar for the simulations; however, the buffer cell simulation distribution shows a much higher velocity magnitude. Most of the cells have a large enough statistical variation to include zero, except for the

buffer cells and those near the first cell. The overlay of the distributions is presented in figure 24.

Figure 24. Overlay of v_z distributions for non-buffer and fixed density buffer cell Couette flows.

The wall velocities were input as ± 0.3 Mach for the non-buffer cell simulation, which results in a speed of $\pm 92.343 \text{ m/s}$. Since only particles impacting the wall have their y velocity component incremented by the wall velocity, the average v_y for the boundary cells is less than the wall velocity by approximately 18%. These results highlight the difference between individual particles having their v_y components modified when colliding with a moving wall and all the particles within a moving buffer cell having their velocities affected. For the buffer cell system, the velocity is greater by about 17% over the non-buffer cell system.

Figure 25. Overlay of v_y distributions for non-buffer and fixed density buffer cell Couette flows.

The velocity in the x -direction also exhibits no unusual behavior. The average cell results are grouped about zero, indicative of the equilibrium condition in this simulation direction.

Figure 26. Overlay of v_x distributions for non-buffer and fixed density buffer cell Couette flows.

However, when computing the average cell velocity using the thermodynamic average velocity (equation 63), the velocity distribution is more within the expected results of zero velocity. The comparison of time averaged instantaneous velocity vs. thermodynamic time averaged velocity is shown in figure 27.

Figure 27. Overlay of average v_z computed using instantaneous and thermodynamic time averaged velocities for Couette flow with fixed density buffer cells.

The probability densities for particles crossing buffer cell boundaries are presented in the next series of figures. All results agree extremely well with theory.

Figure 28. Probability distribution for particles crossing into cell 1.

Figure 29. Probability distribution for particles crossing out of cell 1.

Figure 30. Probability distribution for particles crossing out of cell 21.

Figure 31. Probability distribution for particles crossing into cell 21.

In order to investigate the effect of fixed buffer cell density on the fluid parameters measured, an additional simulation was completed that allowed the density to vary within the buffer cells, but the temperature and v_y in the buffer cells were still fixed at the same values as in the fixed density simulation.

The density results in figure 32 show that the variable density buffer cell simulation compares very well with the non-buffer cell simulation. The density in buffer cell 21 is slightly higher than the corresponding density in the non-buffer cell simulation, but the agreement through the rest of the systems is good.

Figure 32. Overlay of density results for non-buffer cell and variable density buffer cell simulations.

The corresponding temperature results were very similar to the fixed density buffer cell simulation, and are shown in figure 33.

Figure 33. Overlay of temperature results for non-buffer cell and variable density buffer cell simulations.

The largest difference is evident in the plot of the v_z results. In figure 34, the variable density buffer cell results are much closer to the non-buffer cell results.

Figure 34. Overlay of v_z results for non-buffer cell and variable density buffer cell simulations.

The results for the two methods of computing the average velocity are presented in figure 35. The values change somewhat, but not to the extent when the densities are fixed in the buffer cells.

Figure 35. Overlay of velocity results for variable density buffer cell simulations.

5.3 Temperature Gradient Simulation

The last system considered has a temperature gradient instead of the velocity gradient in Couette flow. The gradient is 125 K, centered about 312.5 K. The purpose of this simulation is to compare the steady-state particle densities and temperature distribution for systems without and with buffer cells, in order to quantify the effect of buffer cells on steady-state conditions. In the Couette flow results, the effect of changing every particle's v_y

and most probable velocity (by fixing the buffer cell temperature) had a noticeable effect when compared with the non-buffer cell system results for density and v_z , and less of an impact on the temperature results.

The wall temperatures (250 K and 375 K) for this simulation were chosen so that for the buffer cell system, the buffer cell particle densities would be integral numbers (60 and 40, respectively), and still yield an average density across the system of 50 particles per cell. The temperatures and densities were also selected to keep the pressure at both ends of the system equal so that no pressure gradient would be introduced in the buffer cell system.

The next series of figures are the results comparing the non-buffer cell and buffer cell system. The particle density results are shown in figure 36. The particle densities in the first and last cells are approximately 59 and 43, respectively, for the non-buffer cell simulation.

Figure 36. Particle density results for temperature gradient simulations of non-buffer cell and fixed density buffer cell systems.

The temperature results are presented in figure 37, with the fixed density buffer cell results slightly lower at one end of the system due to the much lower temperature of cell 1.

Figure 37. Temperature distributions for temperature gradient simulations of non-buffer cell and fixed density buffer cell systems.

The average v_z in each cell show the effects of a definite pressure gradient for the fixed density buffer cell system. While the drift appears significant when compared with the non-buffer cell simulation results, the maximum velocity drift of approximately 4.3m/s is still much lower than the average velocity of particles in the system (over 300m/s). These results are shown in figure 38.

Figure 38. The v_z distributions for temperature gradient simulations of non-buffer cell and fixed density buffer cell systems.

While temperature graph shows the gradient, the non-buffer cell system temperature in cell 1 is 262.6 K, not the wall temperature of 250 K, and in cell 20 the temperature is 360.3 K, not 375 K. The fixed density buffer cell system results show some unexpected behavior. The particle density and temperature graphs are accurate reflecting the proper fixing of density and temperature (via most probable velocity) in the fixed density buffer cell

simulation. However, the v_z results show a steady drift of particles in the positive z -direction, conflicting with the equal pressures in the buffer cells. While the other simulations (equilibrium and Couette flow) show excellent agreement with theory, the results for this temperature gradient simulation show some differences due to the pressure gradient evident in cells 2 to 20.

The velocity results for v_y and v_x are in agreement with equilibrium results.

Figure 39. The v_y distributions for temperature gradient simulations of non-buffer cell and fixed density buffer cell systems.

Figure 40. The v_x distributions for temperature gradient simulations of non-buffer cell and fixed density buffer cell systems.

Since the non-buffer cell system has no pressure gradient, it appears that extrapolating the particle density and temperature values to the wall locations (corresponding to the midpoint of the buffer cells) might yield more appropriate results. Because the length of the buffer cell system has been increased by one cell, the average temperatures at the mid-point of the first and last cells in the non-buffer cell system should coincide with the temperatures at the interior boundaries of the buffer cells in the buffer cell system (see figure 11). Setting the temperatures at the mid-points of the buffer cells using equation 28 with the temperatures obtained from the non-buffer cell simulation at the boundaries of the buffer cells should give better results than using the boundary conditions from the non-buffer simulation.

The density results are presented in figure 41 for a system with fixed densities and temperatures of 60 particles and 260.0 K in cell 1, and 43 particles and 363.0 K in cell 21. The results still show some variance with respect to the non-buffer cell results.

Figure 41. Particle densities for non-buffer cell and modified boundary conditions fixed density buffer cell simulations.

The temperature results are similar, with better agreement than the unmodified buffer cell simulation. The temperatures show good agreement throughout the systems. These result are shown in the following figure.

Figure 42. Overlay of temperature profiles for non-buffer cell and modified boundary conditions fixed density buffer cell simulations.

The v_z results show the drift has only been slightly reduced from the unmodified boundary condition simulation velocity drift.

Figure 43. Overlay of v_z profiles for non-buffer cell and modified boundary conditions fixed density buffer cell simulations.

The next series of results show the effect of allowing the density to vary within the buffer cells. The buffer cell temperatures are the same as in the previous set of runs, but by allowing the density to vary, the v_z profile compares very well with the non-buffer cell simulation. The density results, shown in figure 44, show good agreement with the non-buffer cell simulation, except for the buffer cells.

Figure 44. Overlay of density profiles for non-buffer cell and variable density buffer cell simulations.

The temperature profiles show better agreement and are presented in figure 45. The buffer cell average temperatures compare much more favorably with the values obtained using equation 28.

Figure 45. Overlay of temperature profiles for non-buffer cell and variable density buffer cell simulations.

The most dramatic improvement is seen in the v_z profiles, where the velocity drift has been eliminated. The results, shown in figure 46, show excellent agreement with the thermodynamic equilibrium-like conditions expected. However, there is still some difference in the buffer cells that has remain constant throughout the various buffer cell simulations (modified boundary conditions with fixed density and modified boundary conditions variable density).

Figure 46. Overlay of v_z profiles for non-buffer cell and variable density buffer cell simulations.

The results for the boundary crossing velocity distributions for all three temperature gradient buffer cell simulations have similar features. While only the distributions from the variable density buffer cell runs are presented here, the results for the unmodified boundary condition and modified boundary conditions runs are similar.

For the first buffer cell, cell 1, the velocity distributions match theory well, and are presented in figure 47 and figure 48. The velocity distribution of particles crossing into cell 1 from cell 2 is indicative of the higher fluid temperature in cell 2.

Figure 47. Velocity probability distribution for particles crossing into cell 1.

Since the particles in cell 1 are all reset to 260 K (the theory profiles are for $T = 260$ K), the distribution of velocities for particles crossing into cell 2 should overlay with the theoretical distribution curve as can be seen in figure 48.

Figure 48. Probability distribution for particles crossing out of cell 1.

The next two figures show the results for particles crossing into and out of cell 21. These results are similar to those for cell 1, with the cooler fluid temperature in cell 20 resulting in the slightly slower velocity distribution of particles entering cell 21. The theory plot represents a biased Maxwell-Boltzmann distribution with $T = 363$ K.

Figure 49. Probability distribution for particles crossing out of cell 21.

Figure 50. Probability distribution for particles crossing into cell 21.

CHAPTER 6

CONCLUSIONS

The use of buffer cells in DSMC simulations has received considerable attention. The emphasis of these other studies has been in the patching of DSMC-modeled regions with regions modeled by analytic-type methods such as Navier-Stokes. The work presented in this paper has focused on the applicability of using buffer cells to modeling boundary conditions, and on the proper determination of average quantities within buffer cells for patching to regions modeled by other methods.

The use of buffer cells in modeling boundaries was successful for both thermodynamic equilibrium systems and in modeling Couette flow. For the thermodynamic equilibrium case, the density of particles in the buffer cell system was slightly above the average density for the non-buffer cell system, though both should show similar behavior about the theoretical density. For the z -velocity component, a slight pressure gradient is evident in the buffer cell system, but is still quite small. While the average z -component of velocity differed from ideal thermodynamic equilibrium, v_x and v_y for the buffer cell system was more equilibrium-like than the non-buffer cell system, probably due to the removal of any non-zero bias in the buffer cells.

For Couette flow, however, since there is heating of the particles due to the shear force produced by the moving walls and the density is fixed in the end cells, the fixed density buffer cell system's density is lower than the non-buffer cell system. However, the temperature profiles for both systems are very similar, resulting in a larger pressure variation across the buffer cell system. This is apparent in the v_x data, as there is a definite drift in the positive z -direction. However, when the particle density within the buffer

cells is allowed to vary, i.e., not fixed, the velocity drift in the z -direction is eliminated, and the density profile more closely resembles the non-buffer cell density profile. Allowing the density to vary within the buffer cell isolates the buffer cell's boundary condition imitation, showing that indeed buffer cells can model boundary conditions well.

In the temperature gradient simulation, assigning the boundary condition temperatures in the non-buffer cell system to the buffer cell temperatures in the fixed density buffer cell system did not work very well. Although the magnitude of the velocity drift was small compared to the average velocity of particles in the system, there was still a pronounced velocity drift moving particles from one end of the system to the other. Even when the temperature values for the buffer cells were more carefully chosen (modified boundary condition simulation), there was still a velocity drift in the z -direction. It appears that if great care is not taken in choosing the fixed density, the cells that create or destroy particles can set up a steady-state flow where particles are moved across the system. When the buffer cell densities are allowed to vary as in the other system cells, the velocity drift is, for practical purposes, eliminated. However, there is still some odd behavior in the buffer cells, since the v_z 's were not as small as would be expected from the non-buffer cell simulation.

The results of the various simulations also demonstrated that care must be taken when computing average velocities in certain situations. When density was fixed within a buffer cell, the average v_z determined from the instantaneous velocity was markedly different than the other system cells. When computed correctly using instantaneous momentum, the v_z results were much closer to the average v_z in the adjacent non-buffer cells. However, for buffer cell systems where the density was not fixed within the

buffer cells, there was little difference between the two methods for computing average velocity.

This work has shown the suitability of using buffer cells to implement boundary conditions, the care required to model a dilute gas well when fixing the particle density in a buffer cell, and that using momentum to calculate average velocity in a fixed density buffer cell is required. The latter two results are particularly important considerations when using buffer cells to patch a non-DSMC region to a DSMC-modeled region.

BIBLIOGRAPHY

- [1] *Microscopic Simulation of Complex Flows*, M. Mareschal ed., NATO ASI Series, Vol. 236 (Plenum, New York, 1990); *Microscopic Simulations of Complex Hydrodynamic Phenomena*, M. Mareschal and B. L. Holian, Eds., NATO ASI Series, Vol. 292 (Plenum, New York, 1992).
 - [2] E. P. Muntz, "Rarefied Gas Dynamics", *Ann. Rev. Fluid Mech.* 21, 387 (1989).
 - [3] D. A. Erwin, G. C. Pham-Van-Diep and E. P. Muntz, "Nonequilibrium gas flows. I: A detailed validation of Monte Carlo direct simulation for monatomic gases", *Phys. Fluids A* 3, 697 (1991).
 - [4] D. C. Wadsworth, "Slip effects in a confined rarefied gas. I: Temperature slip", *Phys. Fluids A* 5, 1831 (1993).
 - [5] D. C. Wadsworth and D. A. Erwin, "One-Dimensional Hybrid Continuum/Particle Simulation Approach for Rarefied Hypersonic Flows", AIAA-90-1690 (1990).
 - [6] D. A. Anderson, J. C. Tannehill, R. H. Pletcher, *Computational Fluid Mechanics and Heat Transfer*, Chapter 5 (McGraw-Hill, New York, 1984).
 - [7] C. Cercignani, *Mathematical Methods in Kinetic Theory*, (Plenum, New York, 1969); C. Cercignani, *The Boltzmann Equation and its Applications*, (Springer-Verlag, New York, 1988).
 - [8] F. Reif, *Fundamentals of Statistical and Thermal Physics*, (McGraw-Hill, New York, 1965).
 - [9] A. L. Garcia, *Numerical Methods for Physics*, Chapter 10 (Prentice Hall, Englewood Cliffs NJ, 1994).
 - [10] G. A. Bird, *Molecular Gas Dynamics*, (Clarendon, Oxford, 1976).
 - [11] J. A. McLennan, *Introduction to Non-equilibrium Statistical Mechanics*, (Prentice Hall, Englewood Cliffs NJ, 1989).
 - [12] D. J. Tritton, *Physical Fluid Dynamics*, (Clarendon, Oxford, 1988).
-

- [13] M. Holt, *Numerical Methods in Fluid Dynamics*, Chapter 6, (Springer-Verlag, New York, 1977).
- [14] E. M. Murman and J. D. Cole, "Calculation of Plane Steady Transonic Flows", *AIAA J.* **9**, 114 (1971).
- [15] B. J. Alder and T. E. Wainwright, "Phase transition for a hard sphere system", *J. Chem Phys.* **27**, 1208 (1957).
- [16] M. P. Allen and D. J. Tildesley, *Computer Simulation of Liquids*, (Clarendon, Oxford, 1987).
- [17] G. A. Bird, "Approach to translational equilibrium in a rigid sphere gas", *Phys. Fluids A* **6**, 1518 (1963).
- [18] G. A. Bird, "A Contemporary Implementation of The Direct Simulation Monte Carlo Method", in *Microscopic Simulations of Complex Hydrodynamic Phenomena*, M. Mareschal and B. L. Holian, Eds., NATO ASI Series, Vol 292 (Plenum, New York, 1992).
- [19] W. Wagner, "A Convergence Proof for Bird's Direct Simulation Monte Carlo Method for the Boltzmann Equation", *J. Stat. Phys.* **66**, 1011 (1992).
- [20] G. A. Bird, *Molecular Gas Dynamics and the Direct Simulation of Gas Flows*, (Clarendon, Oxford, 1994).
- [21] D. L. Morris, L. Hannon and A. L. Garcia, "Slip Length in a Dilute Gas", *Phys. Rev. A* **46**, 5279 (1992).
- [22] E. Salomons and M. Mareschal, "Usefulness of the Burnett Description of Strong Shock Waves", *Phys. Rev. Lett.* **69**, 269 (1992).

Appendix A

```

#define _DSMCNE
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include "dsmcprod.h"

double x[N_PART], v[N_PART][4],
       ave_n[NCELL+1], ave_u[NCELL+1][4], ave_T[NCELL+1], ave_p[NCELL+1][4],
       sum_n[NCELL+1], sum_v[NCELL+1][4], sum_v2[NCELL+1],
       vcm[4], vrel[4],
       vrmax[NCELL+1], selxtra[NCELL+1], xcell[NCELL+1],
       vwall[3], dverr[3], dvtot[3], force [3], ferr[3], delv[3], xwall[3],
       direction[3], bufcell_xbndry[MAX_BUFFER_CELLS+1][3],
       bufcell_vel[MAX_BUFFER_CELLS+1], bufcell_mpv[MAX_BUFFER_CELLS+1];

int Xref[NCELL+1][N_PART/10], list_part[N_PART/10],
    npart[1], nsamp[1], idum[1],
    cell_n[NCELL+1],
    strikes[3], n_buffercells, flag_thermal,
    buffer_cell[MAX_BUFFER_CELLS+1], n_reset[MAX_BUFFER_CELLS+1],
    bufcell_rho[MAX_BUFFER_CELLS+1],
    bufcell_vel_rxpout[MAX_BUFFER_CELLS+1],
    bufcell_vel_lxpout[MAX_BUFFER_CELLS+1],
    bufcell_vel_rxnout[MAX_BUFFER_CELLS+1],
    bufcell_vel_lxnout[MAX_BUFFER_CELLS+1],
    bufcell_left_bndry[MAX_BUFFER_CELLS+1],
    bufcell_right_bndry[MAX_BUFFER_CELLS+1],
    bufcell_vel_rx[MAX_BUFFER_CELLS+1][41],
    bufcell_vel_lx[MAX_BUFFER_CELLS+1][41];

int main()
{
    FILE *infile, *rstfile, *outfile, *dumpfile;
    static double volume, eff_num, mfp, mpv, vwall_m, conv_m,
               tau, tsamp, vgrad, visc, viscrr, coeff,
               vel_bin[42], bufcell_temp[MAX_BUFFER_CELLS+1];
    static double delta_wall_temp, cell_temp[NCELL+1],
               wall_temp[3], mpvel[NCELL+1], sys_temp, sys_length;
    static double y[N_PART][4], z[N_PART][4];
    static int i, j, jcell, ncell, nstep, istep, col, cont, rho,
               npart2, freset, del_particle, irun, n_runs,
               npartstart, seed, sum_delta_part;
    static char fileout[20], filerestart[20], filedump[20],
               file_irun[20], buffer[5], filein[20];

    double test;

```

```

test = sin(1.);
wall_temp[1] = 0.0;
wall_temp[2] = 0.0;
sum_delta_part = 0;
/* Read in problem parameters */
printf("Enter the name of the startup file: ");
scanf("%s", &filein);
if ((inpf = fopen(filein, "r")))
{
    fscanf(inpf, "%d %d", &rho, &ncell);
    fscanf(inpf, "%lf", &sys_length);
    fscanf(inpf, "%lf", &vwall_m);
    fscanf(inpf, "%d", &flag_thermal);
    if (flag_thermal)
        fscanf(inpf, "%lf %lf", &wall_temp[1], &wall_temp[2]);
    else
        fscanf(inpf, "%lf", &sys_temp);
    fscanf(inpf, "%d", &n_buffercells);
    for (i = 1; i <= n_buffercells; i++)
        fscanf(inpf, "%d", &buffer_cell[i]);
    for (i = 1; i <= n_buffercells; i++)
        fscanf(inpf, "%lf", &bufcell_vel[i]);
    for (i = 1; i <= n_buffercells; i++)
        fscanf(inpf, "%lf", &bufcell_temp[i]);
    fscanf(inpf, "%s", &fileout);
    fscanf(inpf, "%d", &freset);
    fscanf(inpf, "%s", &filerestart);
    fscanf(inpf, "%d", &n_runs);
    fscanf(inpf, "%d", &nstep);
    fscanf(inpf, "%d", &seed);
    fscanf(inpf, "%s", &filedump);
}
else
    exit(1);
if (strlen(fileout) > 7)
{
    printf("Output file base is too long (> 7 charaters).\n");
    exit(1);
}
*npart = rho * ncell;
npartstart = *npart;
printf("The average # of particles per cell is %d\n", rho);
xwall[1] = 0.0;
xwall[2] = sys_length;
/* Setup mid-point location of each cell */
for (i = 1; i <= ncell; i++)
{
    xcell[i] = ((double)i - .5)/(double) ncell * sys_length;
}
printf("xwall[1] is %10.3e and xwall[2] is %10.3e\n", xwall[1], xwall[2]);
volume = sys_length * sys_length * sys_length;

```

/* Initialize floating point for Borland compilers*/
/* Set wall temperatures to zero - specular */
/* wall assumption */

/* Number of particles per cell */
/* and # of cells in system */
/* System length, in meters */
/* Wall velocity for nonlinear */
/* Thermal wall set flag */
/* Read in wall temperatures */

/* Read in system temperature */

/* Number of buffer cells */
/* Read in buffer cells */

/* Read in buffer cell velocities */

/* Read in buffer cell temperatures */

/* Base name of output files */
/* 0 - new run, 1 - resume */

/* File with data to resume */

/* Number of runs (batches) */
/* Number of time steps/run */
/* Seed for these runs */
/* File to store info to allow restart */

/* Number of particles */

/* Location of leftmost boundary */
/* Location of rightmost boundary */

```

    eff_num = (double) (DENSITY/MASS) * volume/((double) *npart);
    printf("Each particle represents %f atoms\n", eff_num);
    mfp = volume/(sqrt(2.)*(double) (PI*DIA*DIAM)*((double) *npart)*eff_num);
    printf("System width is %f mean free paths\n", sys_length/mfp);
/* Set up initial wall velocities */
    for (i = 1; i <= 2; i++)
    {
        if (flag_thermal)
            conv_m = vwall_m * sqrt(5./3.*(double) (BOLTZ*wall_temp[i]/MASS));
        else
            conv_m = vwall_m * sqrt(5./3.*(double) (BOLTZ*sys_temp/MASS));
        if (i == 1)
            vwall[1] = -conv_m;
        else
            vwall[2] = conv_m;
    }
/* Set up system temperature */
    if (flag_thermal)
        sys_temp = (wall_temp[1] + wall_temp[2]) / 2.0;
/* Set up cell most probable velocities and establish temperature gradient */
    delta_wall_temp = wall_temp[2] - wall_temp[1];
    for (i = 1; i <= ncell; i++)
    {
        if (fabs(delta_wall_temp) > EPS)
            cell_temp[i] = delta_wall_temp / (double) (ncell - 1) * (double) i
                + wall_temp[1] - delta_wall_temp / (double) (ncell - 1);
        else
            cell_temp[i] = sys_temp;
        mpvel[i] = sqrt((double) (2.*BOLTZ*cell_temp[i]/MASS));
    }
/* Most probable initial velocity */
    if (fabs(mpvel[1]-mpvel[ncell]) < EPS)
    {
        printf("Most probable velocity is %f\n", mpvel[1]);
    }
    else
    {
        printf("Average most probable velocity across system is: %f\n",
            ((mpvel[1] + mpvel[ncell])/2.0));
    }
/* Set up buffer cell velocities and most probable velocities */
    for (i = 1; i <= n_buffercells; i++)
    {
        conv_m = bufcell_vel[i] * sqrt(5. / 3.*(double) (BOLTZ*bufcell_temp[i]/MASS));
        bufcell_vel[i] = conv_m;
        bufcell_mpv[i] = sqrt((double) (2.*BOLTZ*bufcell_temp[i]/MASS));
        bufcell_rho[i] = (int) floor((double) rho * sys_temp / bufcell_temp[i] + 0.01);
    }
    printf("Wall velocities = %f and %f m/s \n", vwall[1], vwall[2]);
/* Compute left and right boundaries for buffer cells */
    for (i = 1; i <= n_buffercells; i++)

```

```

    {
        bufcell_xbndry[i][1] = xcell[buffer_cell[i]] - xcell[1];
        bufcell_xbndry[i][2] = xcell[buffer_cell[i]] + xcell[1];
    }
    *idum = seed;
    if (!freset)
    {
        /* Assign random positions */
        rand_assign_1d(x, *npart, sys_length, idum);
        /* Assign thermal velocities using Gaussian random numbers */
        rand_assign(y, *npart, 3, (double) 1.0, idum);
        rand_assign(z, *npart, 3, (double) 2.0*PI, idum);
        for (i = 1; i <= *npart; i++)
        {
            jcell = (int) ceil((double) ncell * x[i] / sys_length);
            if (jcell < 1 || jcell > ncell)
            {
                printf("Particle %d is outside system.\n", i);
                exit(1);
            }
            for (j = 1; j <= 3; j++)
            {
                v[i][j] = mpvel[jcell] * sqrt(-log(1.-y[i][j])) * sin(z[i][j]);
            }
        }
        /* Assign velocity gradient */
        for (i = 1; i <= *npart; i++)
        {
            v[i][2] = v[i][2] + vwall[1] + (vwall[2] - vwall[1]) * x[i] / sys_length;
        }
    }
    else
    {
        printf("restart file name is %s\n", filerestart);
        close(inpfile);
        if ((rstfile = fopen(filerestart, "r")))
        {
            fscanf(rstfile, "%d", &sum_delta_part);
            fscanf(rstfile, "%d", npart);
            j = 0;
            for (i = 1; i <= *npart; i++)
            {
                fscanf(rstfile, "%lf,%lf,%lf,%lf", &x[i], &v[i][1], &v[i][2], &v[i][3]);
                j += 1;
            }
        }
        else
            exit(1);
        close(rstfile);
    }
    tau = 0.2 * (sys_length / (double) ncell) / (max(mpvel[1], mpvel[ncell]));

```

/* Time step */

```

printf("The time step is: %14.7e\n", tau);
coeff = 0.5 * eff_num * (double) (PI * DIAM*DIAM) * tau / (volume/((double) ncell));
init_array_1d(vrmax, ncell, (3. * min(mpvel[1], mpvel[ncell])));
sorter(ncell, sys_length);
for (irun = 1; irun <= n_runs; irun++)
{
/* Initialize histogram bins for particles crossing boundaries of cells with fixed density */
for (i = 1; i <= n_buffercells; i++)
{
    bufcell_vel_rxpout[i] = 0; /* Count of particles crossing right */
    bufcell_vel_lxpout[i] = 0; /* or left boundary of buffer cell, */
    bufcell_vel_rxnout[i] = 0; /* but outside histogram velocity bins */
    bufcell_vel_lxnout[i] = 0;
    bufcell_left_bndry[i] = 0; /* Count of particles which fall */
    bufcell_right_bndry[i] = 0; /* exactly on a buffer cell boundary */
    for (j = 1; j <= 40; j++)
    {
        if (irun == 1) /* Histogram: -1000 to 1000 */
            vel_bin[j] = -1050.0 + 50.0 * (double) j;
        bufcell_vel_rx[i][j] = 0; /* Count of particles whose velocities */
        bufcell_vel_lx[i][j] = 0; /* fall within a velocity bin */
    }
}
*nsamp = 0;
tsamp = 0.0;
/* Initialize average quantities */
for (i = 1; i <= ncell; i++)
{
    ave_n[i] = 0.0;
    ave_T[i] = 0.0;
    for (j = 1; j <= 3; j++)
    {
        ave_u[i][j] = 0.0;
        ave_p[i][j] = 0.0;
    }
}
for (i = 1; i <= 2; i++)
{
    dverr[i] = 0.0;
    dvtot[i] = 0.0;
    delv[i] = 0.0;
}
del_particle = *npart;
for (i = 1; i <= n_buffercells; i++)
{
    del_particle += (bufcell_rho[i] - cell_n[buffer_cell[i]]); /* Determine # of particles */
} /* at start of run */
for (istep = 1; istep <= nstep; istep++)
{
    mover(mpvel, tau, ncell);
    sorter(ncell, sys_length);
}

```

```

col = colide(ncell, coeff);

if (istep > nstep/10)                                /* Start taking samples */
{
    sampler(ncell, sys_length);
    for (i = 1; i <= 2; i++)
    {
        dvtot[i] = dvtot[i] + delv[i];
        delv[i] = delv[i] * delv[i];
        dverr[i] = dverr[i] + delv[i];
    }
    tsamp = tsamp + tau;
}
}
sorter(ncell, sys_length);
del_particle -= *npart;
for (i = 1; i <= n_buffercells; i++)
{
    del_particle -= (bufcell_rho[i] - cell_n[buffer_cell[i]]);    /* Determine change in # of */
                                                                    /* particles at end of run */
}
/* Keep running sum of change in # of particles */
for (i = 1; i <= n_buffercells; i++)
{
    for (j = 1; j <= 40; j++)
    {
        if (vel_bin[j] < 0.0)
            sum_delta_part += bufcell_vel_rx[i][j] - bufcell_vel_lx[j][i];
        else
            sum_delta_part += bufcell_vel_lx[i][j] - bufcell_vel_rx[j][i];
    }
    sum_delta_part += bufcell_vel_lxpout[i] - bufcell_vel_rxpout[i]
                    + bufcell_vel_rxnout[i] - bufcell_vel_lxnout[i];
}
sum_delta_part -= del_particle;
/* Normalize statistics */
for (i = 1; i <= ncell; i++)
{
    ave_n[i] = ave_n[i] / (double) *nsamp;
    ave_T[i] = ave_T[i] * (double) (MASS/(3.*BOLTZ*(double) *nsamp));
    for (j = 1; j <= 3; j++)
    {
        ave_u[i][j] = ave_u[i][j] / (double) *nsamp;
        ave_p[i][j] = ave_p[i][j] / (double) *nsamp / ave_n[i];
    }
}
for (i = 1; i <= 2; i++)
{
    dverr[i] = dverr[i] / (double) (*nsamp - 1);
}
for (i = 1; i <= 2; i++)
{

```

```

    dverr[i] = dverr[i] - dvtot[i]*dvtot[i] / (double) *nsamp/(double) *nsamp;
}
for (i = 1; i <= 2; i++)
{
    dverr[i] = sqrt(dverr[i] * (double) *nsamp);
}
for (i = 1; i <= 2; i++)
{
    force[i] = eff_num * dvtot[i] * (double) (MASS/tsamp/sys_length/sys_length);
    ferr[i] = eff_num * dverr[i] * (double) (MASS/tsamp/sys_length/sys_length);
}
vgrad = (vwall[2] - vwall[1])/sys_length;
if (vgrad != 0)
{
    visc = 0.5 * (-force[1] + force[2])/vgrad;
    viscerr = 0.5 * (ferr[1] + ferr[2]) / vgrad;
    printf("viscosity is %10.3e, +/- %10.3e\n", visc, viscerr);
}
strcpy(file_irun, fileout);
strcat(file_irun, gcvt((double) irun, 3, buffer));
strcat(file_irun, ".thm");
printf("Output file name #%d is %s\n", irun, file_irun);
if ((outfile = fopen(file_irun, "w")))
{
    fprintf(outfile, "Initial seed is %d\n", seed);
    for (i = 1; i <= ncell; i++)
    {
        fprintf(outfile, "%14.7e,%14.7e,%14.7e,%14.7e,%14.7e,%14.7e\n",
            xcell[i], ave_n[i], ave_p[i][1], ave_p[i][2], ave_p[i][3],
            ave_T[i]);
    }
}
/* Write out boundary crossing info (count vs. velocity histogram) */
for (i = 1; i <= n_buffercells; i++)
{
    fprintf(outfile, "Buffer cell %d: Bndry Velocity Histogram\n",
        buffer_cell[i]);
    for (j = 1; j <= 40; j++)
    {
        fprintf(outfile, "%lf,%d,%d\n", vel_bin[j], bufcell_vel_lx[i][j],
            bufcell_vel_rx[i][j]);
    }
    fprintf(outfile, "%d,%d\n", bufcell_vel_lxnout[i],
        bufcell_vel_rxnout[i]);
    fprintf(outfile, "%d,%d\n", bufcell_vel_lxpout[i],
        bufcell_vel_rxpout[i]);
    fprintf(outfile, "%d,%d\n", bufcell_left_bndry[i],
        bufcell_right_bndry[i]);
    fprintf(outfile, "Final # in cell: %d\n", cell_n[buffer_cell[i]]);
}
fprintf(outfile, "The change in particles is %d\n", del_particle);
}

```

```

fclose(outfile);
strcpy(file_irun, fileout);
strcat(file_irun, gcvt((double) irun, 3, buffer));
strcat(file_irun, ".out");
printf("Output file name #%d is %s\n", irun, file_irun);
if ((outfile = fopen(file_irun, "w")))
{
    fprintf(outfile, "Initial seed is %d\n", seed);
    for (i = 1; i <= ncell; i++)
    {
        fprintf(outfile, "%14.7e,%14.7e,%14.7e,%14.7e,%14.7e,%14.7e\n",
            xcell[i], ave_n[i], ave_u[i][1], ave_u[i][2], ave_u[i][3],
            ave_T[i]);
    }
}
/* Write out boundary crossing info (count vs. velocity histogram) */
for (i = 1; i <= n_buffercells; i++)
{
    fprintf(outfile, "Buffer cell %d: Bndry Velocity Histogram\n",
        buffer_cell[i]);
    for (j = 1; j <= 40; j++)
    {
        fprintf(outfile, "%lf,%d,%d\n", vel_bin[j], bufcell_vel_lx[i][j],
            bufcell_vel_rx[i][j]);
    }
    fprintf(outfile, "%d,%d\n", bufcell_vel_lxnout[i],
        bufcell_vel_rxnout[i]);
    fprintf(outfile, "%d,%d\n", bufcell_vel_lxpout[i],
        bufcell_vel_rxpout[i]);
    fprintf(outfile, "%d,%d\n", bufcell_left_bndry[i],
        bufcell_right_bndry[i]);
    fprintf(outfile, "Final # in cell: %d\n", cell_n[buffer_cell[i]]);
}
fprintf(outfile, "The change in particles is %d\n", del_particle);
}
fclose(outfile);
if ((dumpfile = fopen(filedump, "w")))
{
    fprintf(dumpfile, "%d\n", sum_delta_part);
    fprintf(dumpfile, "%d\n", *npart);
    for (i = 1; i <= *npart; i++)
    {
        fprintf(dumpfile, "%14.7e,%14.7e,%14.7e,%14.7e\n",
            x[i], v[i][1], v[i][2], v[i][3]);
    }
}
fclose(dumpfile);
}
return(0);
}

```



```

#define _NOTDSMCNE
#define PI 3.1415926

#include <math.h>
#include <stdlib.h>
#include <stdio.h>
#include "dsmcprod.h"

void sampler (int ncell, double len)
{
    int ipart, jcell, i, j, cont;

    for (i = 1; i <= ncell; i++)
    {
        sum_n[i] = 0.0;
        sum_v2[i] = 0.0;
        for (j = 1; j <= 3; j++)
        {
            sum_v[i][j] = 0.0;
        }
    }
    for (ipart = 1; ipart <= *npart; ipart++)
    {
        jcell = (int) ceil((double) ncell * x[ipart]/len);
        sum_n[jcell] += 1.0;
        sum_v2[jcell] += v[ipart][1] * v[ipart][1] + v[ipart][2] * v[ipart][2] + v[ipart][3] * v[ipart][3];
        for (i = 1; i <= 3; i++)
        {
            sum_v[jcell][i] += v[ipart][i];
        }
    }
    for (i = 1; i <= ncell; i++)
    {
        ave_n[i] += sum_n[i];
        for (j = 1; j <= 3; j++)
        {
            ave_p[i][j] = ave_p[i][j] + sum_v[i][j];
            sum_v[i][j] = sum_v[i][j] / sum_n[i];
            ave_u[i][j] = ave_u[i][j] + sum_v[i][j];
        }
        ave_T[i] = ave_T[i] + (sum_v2[i] / sum_n[i] -
            (sum_v[i][1]*sum_v[i][1] + sum_v[i][2]*sum_v[i][2] +
            sum_v[i][3]*sum_v[i][3])) * (1 + 1 / sum_n[i]));
    }
    *nsamp = *nsamp + 1;
    return;
}

```

```

int colide(int ncell, double coeff)
{
    double select, crm, cr, cos_th, sin_th, phi;
    int jcell, number, nsel, isel, k, kk, ip1, ip2, i, cont;
    static int col = 0;

    for (jcell = 1; jcell <= ncell; jcell++)
    {
        number = cell_n[jcell];
        if (number > 1)
        {
            select = coeff * (double) (number * number) * vmax[jcell]
                + selxtra[jcell];
            nsel = (int) floor(select);
            selxtra[jcell] = select - (double) nsel;
            crm = vmax[jcell];
            for (isel = 1; isel <= nsel; isel++)
            {
                k = (int) floor((double) ran1(idum)*(double) number);
                kk = (int) fmod(ceil((double) k + (double) ran1(idum) *
                    (double) (number-1)), (double) number);
                ip1 = Xref[jcell][k];
                ip2 = Xref[jcell][kk];
                if ( ip1 > *npart || ip2 > *npart )
                {
                    printf("Either ip1 or ip2 is greater than npart\n");
                    exit(1);
                }
                if ( ip1 < 1 || ip2 < 1 )
                {
                    printf("Either ip1 or ip2 is less than 1\n");
                    exit(1);
                }
            }
            cr = norm_vectors(3, ip1, ip2, v);
            if (cr > crm)
                crm = cr;
            if (cr/vmax[jcell] > (double) ran1(idum))
            {
                col = col + 1;
                for (i = 1; i <= 3; i++)
                {
                    vcm[i] = 0.5 * (v[ip1][i] + v[ip2][i]);
                }
                cos_th = 1. - 2. * (double) ran1(idum);
                sin_th = sqrt(1. - cos_th * cos_th);
                phi = 2. * PI * (double) ran1(idum);
                vrel[1] = cr * cos_th;
                vrel[2] = cr * sin_th * cos(phi);
                vrel[3] = cr * sin_th * sin(phi);
                for (i = 1; i <= 3; i++)
                {

```

```

        v[ip1][i] = vcm[i] + 0.5 * vrel[i];
        v[ip2][i] = vcm[i] - 0.5 * vrel[i];
    }
}
}
vrmax[jcell] = crm;
}
}
return(col);
}

void sorter(int ncell, double len)
{
    int i, jcell, j;

    for (i = 1; i <= ncell; i++)
    {
        cell_n[i] = 0;
    }
    for (i = 1; i <= *npart; i++)
    {
        jcell = (int) ceil((double) (ncell * x[i] / len));
        if (jcell > ncell || jcell < 1)
        {
            printf("x[%d] is greater than %d or less than 1\n", i, ncell);
            exit(1);
        }
        Xreff[jcell][cell_n[jcell]] = i;
        cell_n[jcell] += 1;
    }
    return;
}

void reset(int index, int num_reset, int ncell, double len)
{
    int i, j, last_entry, search_index, flag, temp, add_index, move_index, inc, vv;
    static double y[N_PART/3][4], z[N_PART/3][4];

    last_entry = *npart;
    if (num_reset <= bufcell_rho[index])
    {
        rand_assign(y, num_reset, 3, (double) 1.0, idum);
        rand_assign(z, num_reset, 3, ((double) 2.0*PI), idum);
        for (i = 1; i <= num_reset; i++)
        {
            /* Assign random positions */
            x[list_part[i]] = ((double) (buffer_cell[index] - 1) + ran1(idum)) * len / (double) ncell;
            /* Assign thermal velocities using Gaussian random numbers */
            for (j = 1; j <= 3; j++)
            {
                v[list_part[i]][j] = bufcell_mpv[index] * sqrt(-log(1.-y[i][j]))

```

```

        * sin(z[i][j]);
    if (j == 2)
        v[list_part[i][j]] += bufcell_vel[index];
    }
}
rand_assign(y, (bufcell_rho[index]-num_reset+1), 3, (double) 1.0, idum);
rand_assign(z, (bufcell_rho[index]-num_reset+1), 3, (double) 2.0*PI, idum);
for (add_index = 1; add_index <= (bufcell_rho[index] - num_reset); add_index++)
{
/* Assign Block */
/* Assign random positions */
    x[*npart+add_index] = ((double) (buffer_cell[index] - 1) + ran1(idum))
        * len / (double) ncell;
/* Assign thermal velocities using Gaussian random numbers */
    for (j = 1; j <= 3; j++)
    {
        v[*npart+add_index][j] = bufcell_mpv[index] * sqrt(-log(1.-y[add_index][j]))
            * sin(z[add_index][j]);
        if (j == 2)
            v[*npart+add_index][j] += bufcell_vel[index];
    }
}
*npart = *npart + (bufcell_rho[index] - num_reset);
}
else
{
/* Assign Block */
    rand_assign(y, bufcell_rho[index], 3, (double) 1.0, idum);
    rand_assign(z, bufcell_rho[index], 3, ((double) 2.0*PI), idum);
    for (i = 1; i <= bufcell_rho[index]; i++)
    {
/* Assign random positions */
        x[list_part[i]] = ((double) (buffer_cell[index] - 1) + ran1(idum)) * len / (double) ncell;
/* Assign thermal velocities using Gaussian random numbers */
        for (j = 1; j <= 3; j++)
        {
            v[list_part[i][j]] = bufcell_mpv[index] * sqrt(-log(1.-y[i][j]))
                * sin(z[i][j]);
            if (j == 2)
                v[list_part[i][j]] += bufcell_vel[index];
        }
    }
    for (move_index = num_reset; move_index > bufcell_rho[index]; move_index--)
    {
/* move particle while decrementing npart */
        if (list_part[move_index] != last_entry)
        {
            x[list_part[move_index]] = x[last_entry];
            for (j = 1; j <= 3; j++)
            {
                v[list_part[move_index][j]] = v[last_entry][j];
            }
        }
    }
}

```

```

    }
  }
  last_entry -= 1;
}
*npart = last_entry;

}
return;
}

void partial_reset(int index, int num_reset, int ncell, double len)
{
  int i, j, last_entry, search_index, flag, temp, add_index, move_index,
  inc, vv;
  static double y[N_PART/3][4], z[N_PART/3][4];

  rand_assign(y, num_reset, 3, (double) 1.0, idum);
  rand_assign(z, num_reset, 3, ((double) 2.0*PI), idum);
  for (i = 1; i <= num_reset; i++)
  {
    /* Assign random positions */
    x[list_part[i]] = ((double) (buffer_cell[index] - 1) + ran1(idum))
      * len / (double) ncell;
    /* Assign thermal velocities using Gaussian random numbers */
    for (j = 1; j <= 3; j++)
    {
      v[list_part[i]][j] = bufcell_mpv[index] * sqrt(-log(1.-y[i][j]))
        * sin(z[i][j]);
      if (j == 2)
        v[list_part[i]][j] += bufcell_vel[index];
    }
  }
}

int cell_list(int index, int ncell, double len)
{
  int i;

  if (index != 1)
    sorter(ncell, (double) len);
  for (i = 0; i < cell_n[buffer_cell[index]]; i++)
    list_part[i+1] = Xref[buffer_cell[index]][i];
  return(cell_n[buffer_cell[index]]);
}

void histo(int ipart, double x_old, int ibufcell, double xbndry[][3])
{
  int index;

  /* Check for left boundary crossing first */
  if (xbndry[ibufcell][1] > (xwall[1] + EPS))

```

```

{
  if (((x_old - xbndry[ibufcell][1])*(x[ipart] - xbndry[ibufcell][1])) < 0.0)
  {
    index = (int) ceil((double) ((v[ipart][1]+1000.0)/50.0));
    if (index > 0 && index < 41)
      bufcell_vel_lx[ibufcell][index] += 1;
    else
    {
      if (v[ipart][1] > 0.0)
        bufcell_vel_lxpout[ibufcell] += 1;
      if (v[ipart][1] < 0.0)
        bufcell_vel_lxnout[ibufcell] += 1;
    }
  }
}
if ((x_old - xbndry[ibufcell][1]) == 0.0 || (x[ipart] - xbndry[ibufcell][1]) == 0.0)
  bufcell_left_bndry[ibufcell] += 1;
/* Checking for right boundary crossing of particle i */
if (xbndry[ibufcell][2] < (xwall[2] - EPS))
{
  if (((x_old - xbndry[ibufcell][2])*(x[ipart] - xbndry[ibufcell][2])) < 0.0)
  {
    index = (int) ceil((double) ((v[ipart][1]+1000.0)/50.0));
    if (index > 0 && index < 41)
      bufcell_vel_rx[ibufcell][index] += 1;
    else
    {
      if (v[ipart][1] > 0.0)
        bufcell_vel_rxpout[ibufcell] += 1;
      if (v[ipart][1] < 0.0)
        bufcell_vel_rxnout[ibufcell] += 1;
    }
  }
}
}
if ((x_old - xbndry[ibufcell][2]) == 0.0 || (x[ipart] - xbndry[ibufcell][2]) == 0.0)
  bufcell_right_bndry[ibufcell] += 1;
return;
}

```

```

void mover(double *mpvel, double tau, int ncell)
{
    double phi, v_parll, dtr, x_old, wall_mpv[3];
    int flag, i, j, cont, ncell_reset;

    strikes[1] = 0;
    strikes[2] = 0;
    delv[1] = 0.0;
    delv[2] = 0.0;
    direction[1] = 1.;
    direction[2] = -1.;
    wall_mpv[1] = mpvel[1];
    wall_mpv[2] = mpvel[ncell];
    if (n_buffercells > 0)
    {
        for(i = 1; i <= n_buffercells; i++)
        {
            n_reset[i] = cell_list(i, ncell, xwall[2]);
            if (bufcell_rho[i] > 0)
                reset(i, n_reset[i], ncell, xwall[2]);
            else
                partial_reset(i, n_reset[i], ncell, xwall[2]);
        }
    }
    for(i = 1; i <= *npart; i++)
    {
        x_old = x[i];
        x[i] = x_old + v[i][1] * tau;
        for (j = 1; j <= n_buffercells; j++)
            histo(i, x_old, j, bufcell_xbndry);
        flag = 0;
        if (x[i] <= xwall[1]) flag = 1;
        if (x[i] >= xwall[2]) flag = 2;
        if (flag > 0)
        {
            strikes[flag] = strikes[flag] + 1;
            delv[flag] = delv[flag] - v[i][2];
            if (flag_thermal == 1)
            {
                v[i][1] = direction[flag] * sqrt(-log(1.0 - (double) ran1(idum))) * wall_mpv[flag];
                phi = 2. * (double) PI * (double) ran1(idum);
                v_parll = sqrt(-log(1.0 - (double) ran1(idum))) * wall_mpv[flag];
                v[i][2] = v_parll * sin(phi) + vwall[flag];
                v[i][3] = v_parll * cos(phi);
            }
            else
                v[i][1] *= -1.0;
            dtr = tau * (x[i] - xwall[flag]) / (x[i] - x_old);
            if (dtr < 0.0)
            {
                printf("dtr is %f\n", dtr);
            }
        }
    }
}
/* buffer cell histogram */

```

```
    printf("x is %f, x_old is %f, v[1] is %f \n", x[i], x_old, v[i][1]);
    exit(1);
}
x[i] = xwall[flag] + v[i][1] * dtr;
delv[flag] = delv[flag] + v[i][2];
if (x[i] < xwall[1] || x[i] > xwall[2])
{
    printf("x[%d] is outside problem space at %f\n", i, x[i]);
    exit(1);
}
}
}
return;
}
```



```

#define M1 259200
#define IA1 7141
#define IC1 54773
#define RM1 (1.0/M1)
#define M2 134456
#define IA2 8121
#define IC2 28411
#define RM2 (1.0/M2)
#define M3 243000
#define IA3 4561
#define IC3 51349
#define EPS 1.2e-12
#define RNMX 1.0-EPS
#include <math.h>

```

```

void init_array(double **arr, int nrow, int ncol, double value)
{
    int    i, j;

    for (i = 1; i <= nrow; i++)
    {
        for (j = 1; j <= ncol; j++)
        {
            arr[i][j] = value;
        }
    }
    return;
}

```

```

void init_harray(double arr[][4], int nrow, int ncol, double value)
{
    int    i, j;

    for (i = 1; i <= nrow; i++)
    {
        for (j = 1; j <= ncol; j++)
        {
            arr[i][j] = value;
        }
    }
    return;
}

```

```

void init_array_1d(double *arr, int nrow, double value)
{
    int    i;

    for (i = 1; i <= nrow; i++)
    {
        arr[i] = value;
    }
    return;
}

```

```

void init_int_array_1d(int *arr, int nrow, int value)
{
    int    i, cont;

    for (i = 1; i <= nrow; i++)
    {
        arr[i] = value;
    }
    return;
}

```

```

void init_int_harray_1d(int *arr, int nrow, int value)
{
    int    i, cont;

    for (i = 1; i <= nrow; i++)
    {
        arr[i] = value;
    }
    return;
}

```

```

double ran1(int *idum)
{
    static  long ix1, ix2, ix3;
    static  double r[98];
    double temp;
    static  int iff = 0;
    int     j;

    if (*idum < 0 || iff == 0)
    {
        iff = 1;
        ix1 = (IC1 - (*idum)) % M1;
        ix1 = (IA1 * ix1 + IC1) % M1;
        ix2 = ix1 % M2;
        ix1 = (IA1 * ix1 + IC1) % M1;
        ix3 = ix1 % M3;
        for (j = 1; j <= 97; j++)
        {

```

```

        ix1 = (IA1 * ix1 + IC1) % M1;
        ix2 = (IA2 * ix2 + IC2) % M2;
        r[j] = (ix1 + ix2 * RM2) * RM1;
    }
    *idum = 1;
}
ix1 = (IA1 * ix1 + IC1) % M1;
ix2 = (IA2 * ix2 + IC2) % M2;
ix3 = (IA3 * ix3 + IC3) % M3;
j = 1 + (97 * ix3) / M3;
if (j > 97 || j < 1)
    printf("Ran1: This cannot happen.\n");
temp = r[j];
r[j] = (ix1 + ix2 * RM2) * RM1;
if (temp > RNMX) return(RNMX);
else return(temp);
}

double norm_vectors(int ncol, int r_index1, int r_index2, double arr[][4])
{
    int i, j;
    double delta, sum, norm;

    sum = 0.;
    for (i = 1; i <= ncol; i++)
    {
        delta = arr[r_index1][i] - arr[r_index2][i];
        sum = sum + delta * delta;
    }
    norm = sqrt(sum);
    return(norm);
}

void rand_assign(double arr[][4], int nrow, int ncol, double len, int *idum)
{
    int i, j;
    double ran_num;

    for (i = 1; i <= nrow; i++)
    {
        for (j = 1; j <= ncol; j++)
        {
            ran_num = ran1(idum);
            arr[i][j] = ran_num * len;
        }
    }
    return;
}

```

```
void rand_assign_1d(double *arr, int nrow, double len, int *idum)
{
    int    i, j;
    double ran_num;

    for (i = 1; i <= nrow; i++)
    {
        ran_num = ran1(idum);
        arr[i] = ran_num * len;
    }
    return;
}
```