1992

The effects of a biased collision rule on viscosity in a dilute gas

Kevin P. Siegel
San Jose State University

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

Recommended Citation

Siegel, Kevin P., "The effects of a biased collision rule on viscosity in a dilute gas" (1992). *Master's Theses*. 507.

DOI: <https://doi.org/10.31979/etd.7e7k-ujzt>
https://scholarworks.sjsu.edu/etd_theses/507

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.

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 1351078

The effects of a biased collision rule on viscosity in a dilute gas

Siegel, Kevin Paul, M.S.

San Jose State University, 1992

U·M·I

300 N. Zeeb Rd.
Ann Arbor, MI 48106

THE EFFECTS OF A BIASED
COLLISION RULE ON VISCOSITY IN A DILUTE GAS

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

Kevin P. Siegel

December, 1992

APPROVED FOR THE DEPARTMENT OF PHYSICS

A handwritten signature in cursive script, reading "Alejandro Garcia", positioned above a horizontal line.

Dr. Alejandro Garcia

A handwritten signature in cursive script, reading "Berni Alder", positioned above a horizontal line.

Dr. Berni Alder

A handwritten signature in cursive script, reading "Patrick Hamill", positioned above a horizontal line.

Dr. Patrick Hamill

APPROVED FOR THE UNIVERSITY

A handwritten signature in cursive script, reading "Serena M. Stanford", positioned above a horizontal line.

ABSTRACT

THE EFFECTS OF A BIASED
COLLISION RULE ON VISCOSITY IN A DILUTE GAS

by Kevin P. Siegel

In this thesis, the effect that a biased collision rule has on reducing viscosity was studied using Direct Simulation Monte Carlo (DSMC) for a dilute gas. Viscosity was measured in two flows—Poiseuille and Rayleigh-Bénard—each composed of Helium atoms at realistic temperatures and densities. Both the normal DSMC collision algorithm and a biased collision rule were simulated. In normal DSMC, the exit velocities of the colliding particles are distributed in random directions. Using the biased collision rule, these velocities are aligned with the center of mass velocity. This collision rule is called “go with the flow” (GWTF) due to the tendency of the post-collision velocities to line up with the stream flow.

Intermixing a small proportion of GWTF collisions with normal collisions significantly lowered the viscosity of the system. However, compared with the normal DSMC algorithm, significant physical deviations were observed.

PREFACE

Background

There are many types of algorithms used for simulating fluid dynamics. Particle methods (e.g., Molecular Dynamics) repeatedly calculate the dynamics of a large number of molecules. In these algorithms, the fluid flow of a system is simulated by tracking the position, velocity and collision interactions of every particle. The system is modeled by solving a simple set of ordinary differential equations (the Newtonian dynamics) using small but finite time steps. Unfortunately, for many realistic applications this frequently is computationally expensive (both time and memory intensive).

In the special case of dilute gases, it is possible to make a simplifying assumption. In a dilute gas the probability of more than two particles colliding together within the same time sample is very small. Therefore, if we assume only two body (or binary) collisions, a probabilistic approach using Boltzmann statistics can replace the detailed trajectory calculations in the classical Molecular Dynamics algorithm. This method, first explored by Bird (circa 1960), is called Direct Simulation Monte Carlo (DSMC). While Molecular Dynamics determines particle collisions based on spatial proximity, DSMC randomly picks two particles in a given space and computes the probability of their collision based on their relative velocities. Due to the stochastic nature of the collision calculations, each DSMC particle can represent a large number of molecules in the physical system. With DSMC it becomes possible to model a macroscopically large system with a smaller representative number of particles.

Present Work

Recent work has focused on the computational modeling of turbulent flow. But in order for turbulence to develop in a particle system, a fluid must have at least 10^{10} particles. This is currently even beyond the reach of today's super computers. However, it may still be possible to model turbulence. By artificially lowering the viscosity of a

fluid and effectively raising the system's Reynolds number, turbulence may be induced in systems having significantly fewer particles. Therefore in this work we explore the reduction of a realistic gas system's viscosity using an augmented DSMC collision rule (without violating the fundamental conservation laws).

In order to study the effective viscosity changes induced by an augmented particle collision rule, two dilute gas systems were simulated—Poiseuille and Rayleigh-Bénard. Each system was composed of Helium type particles and simulated under standard conditions (realistic temperatures, densities, etc.). The only variation made to the DSMC algorithm was the inclusion of a new collision rule. In ordinary collisions, the exit velocities of the colliding particles are distributed in random directions. Using the new collision rule, the exit velocities of the colliding particles are aligned with the center of mass velocity of the pair. This collision rule is called “go with the flow” (GWTF) due to the tendency (i.e., bias) of the post-collision velocities to line up with the stream flow.

It was found that intermixing a small proportion of GWTF collisions with normal collisions significantly lowered the effective viscosity of the system. However, significant physical deviations were observed in the typical fluid flows studied (compared with the normal DSMC algorithm).

TABLE OF CONTENTS

	Page
Preface	iv
List of Tables	vii
List of Figures	viii
Chapter	
1. Kinetic Gas Theory	1
General Parameters	1
Boundary Conditions	11
Simple Flows	15
2. Computational Methods	21
Direct Equations	21
Direct Particle Simulation	26
Method Comparison For Dilute Gases	33
3. DSMC Simulations	35
Computational Resources	35
Simulation Structure	37
Flow Configurations	47
4. Simulation Results	51
Poiseuille Flow	51
Rayleigh-Bénard Flow	58
Discussion	69
5. Conclusions	73
Bibliography	77
Appendix	78

TABLES

Table	Page
3.1. DSMC parameters used in dilute gas simulations.	38
4.1. The computed viscosity for Poiseuille runs.	54

FIGURES

Figure	Page
1.1. Kac's simple particle cell model.	2
1.2. The Maxwell-Boltzmann molecular speed distribution for nitrogen gas particles at 273K.	3
1.3. A particle sweeping out a cylindrical volume as it transverses a static particle field.	6
1.4. The continuum regime for air as a function of density and characteristic length.	7
1.5. Thermalization of a particle colliding with a wall and the subsequent velocity distributions.	13
1.6. Flow within one mean free path of a thermal boundary.	14
1.7. A planar Poiseuille system.	16
1.8. The velocity profile of a typical Poiseuille cross-section.	17
1.9. A two-dimensional Rayleigh-Bénard system.	19
2.1. A two-dimensional particle system partitioned into equally spaced cells.	27
3.1. Flow diagram of the three sub-program implementation of DSMC.	36
3.2. The relative velocity and center of mass velocity of a particle pair.	46
3.3. The exit velocities of a particle pair after a normal collision . . .	46
3.4. The exit velocities of a particle pair after a GWTF collision. . .	47
4.1. Three Poiseuille runs approaching steady state stream velocity. .	52
4.2. The steady state, spatially averaged velocity swaths of three Poiseuille simulation runs under various collision rule combinations.	53
4.3. The steady state, spatially averaged temperature swath of a normal rule Poiseuille run.	55
4.4. The steady state, spatially averaged density swath of a normal rule Poiseuille run.	56
4.5. The steady state, spatially averaged temperature swath of a 1% GWTF Poiseuille run.	56

4.6.	The steady state, spatially averaged density swath of a 1% GWTF Poiseuille run.	57
4.7.	The steady state convection flow of a 2-dimensional Rayleigh-Bénard simulation using normal collision rules and a temperature gradient of 455K.	59
4.8.	The x velocity component of Figure 4.7.	60
4.9.	The y velocity component of Figure 4.7.	60
4.10.	The steady state convection flow of a 2-dimensional Rayleigh-Bénard simulation using 1.0% GWTF rules and a temperature gradient of 136.5K.	61
4.11.	The x velocity component of Figure 4.10.	61
4.12.	The y velocity component of Figure 4.10.	62
4.13.	The temporal convergence of the two Rayleigh-Bénard simulation runs, shown in Figures 4.7 and 4.10, to steady state.	63
4.14.	A convection phase diagram showing the average convection velocity versus the applied temperature gradient of two 2-dimensional Rayleigh-Bénard simulation systems.	64
4.15.	The steady state velocity convection flow of a 2-dimensional Rayleigh-Bénard Simulation using normal rules and a temperature gradient of 546K.	65
4.16.	The steady state velocity convection flow of a 2-dimensional Rayleigh-Bénard Simulation using 1.0% GWTF rules and a temperature gradient of 546K.	66
4.17.	The temperature contours of the normal rule Rayleigh-Bénard convection system show in Figure 4.7.	67
4.18.	The density contours of the normal rule Rayleigh-Bénard convection system shown in Figure 4.7.	67
4.19.	The temperature contours of the 1% GWTF rule Rayleigh-Bénard convection system shown in Figure 4.10.	68
4.20.	The density contours of the 1% GWTF rule Rayleigh-Bénard convection system show in Figure 4.10.	69

CHAPTER 1
KINETIC GAS THEORY
General Parameters

Dilute Gases

Simple gases are described by four macroscopic and measurable properties: temperature, pressure, volume and density. In a single species gas, the mass of each particle is identical; therefore, it is equivalent to express density (mass per unit volume) as the total number of particles per unit volume. This mass-less term is often referred to as the system's *number density*, n ; it is often used in fluid dynamics as a parametric criterion for simplifying particle based gas models.

A simple gas is considered dilute if the molecular spacing, δ , between particles in the system is large with respect to the particle's effective diameter, d_{eff} .

$$\sqrt[3]{1/n} = \delta \gg d_{eff} \quad (1.1)$$

The effective diameter in this case is defined by some particle-to-particle interaction potential. This potential can be experimentally derived or theoretically modeled (e.g., assuming hard spheres, the effective diameter is equivalent to the particle's diameter).

Because of the sparse densities involved, dilute gases have simplified particle dynamics and encourage the use of simple models. Many particle collisions regularly occur in a dilute gas (e.g., the collision rate of air under standard conditions is $9.8 \times 10^{24} \text{ m}^{-3}\text{s}^{-1}$). However, the probability of more than two particles being involved in a single collision event is extremely low. It is therefore reasonable to assume two-body (or binary) collisions occur exclusively. This single assumption for dilute gases not only simplifies the collision dynamics involved, but also allows it to be modeled using a

probabilistic approach. By strictly assuming binary particle collisions, we can now make use of kinetic theory to construct a stochastic computational algorithm.

The Kac Model

In 1959 the mathematician Kac proposed a simple particle model for studying a dilute gas under homogeneous conditions.^{1,2} Kac's model consisted of a single spatial cell, free of any external forces, with purely periodic boundaries. Initially the particles have an arbitrary velocity distribution. Since the boundaries in the cell are periodic, the system is both homogenous and isotropic and actual particle positions can be ignored. Also, since the particles are assumed ideal, all energy is kinetic and is intrinsic to the particle's velocity (i.e., $KE = 1/2 mv^2$).

Figure 1.1. Kac's simple particle cell model.

Kac was ultimately interested in constructing the probability P_{ij} , the probability that particles i and j will collide. Intuitively, the faster a particle travels the more likely it is to hit something. Kac assumed that this probability must be a function of the relative velocities of the particles,

¹M. Kac, Probability Theory and Related Topics in Physical Science, (Wiley-Interscience, New York, 1959).

²J. Logan and M. Kac, Phys. Rev. A 13, 458 (1976).

$$P_{ij} = \frac{f(|v_i - v_j|)}{\sum_{ij} f(|v_i - v_j|)} \quad (1.2)$$

where $|v_i - v_j|$ is the relative velocity. As the particles interact they approach equilibrium; Kac mathematically proved Boltzmann's H-theorem for this simple system. The H-theorem states that a distribution of particle velocities approaches the Maxwell-Boltzmann distribution as the system equilibrates.

The Maxwell-Boltzmann distribution of molecular speed, v , is given by,

$$P(v)dv = 4\pi \left(\frac{m}{2\pi kT} \right)^{3/2} v^2 e^{-\frac{1}{2}mv^2/kT} dv \quad (1.3)$$

where $P(v)dv$ is the probability that a particle's speed is between v and $v + dv$, m is the molecular mass, T is the temperature, and k is the Boltzmann constant (a typical distribution for nitrogen at 273K is shown in Figure 1.2).

Figure 1.2. The Maxwell-Boltzmann molecular speed distribution for nitrogen gas particles at 273K. Note that the vertical line indicates the average speed.

Boltzmann Equation

The Boltzmann equation, first proposed by Boltzmann in 1872, describes the dynamics of systems with predominately binary collisions (i.e., dilute gas). Given the statistical velocity distribution, $f(\mathbf{c}, \mathbf{r}, t)$, the Boltzmann equation expresses the change in this distribution. for all time and space.³

The Boltzmann equation can be broken down into the fundamental processes which contribute to velocity distribution changes. These three processes are: free transport (particle movement), external influences (forces), and particle-particle interactions (collisions).

The change of the distribution due to particle movement throughout space (equation 1.4), can be expressed as the dot product of the velocity class $\mathbf{c}$ (the velocities of the particles), with the spatial gradient of the distribution.

$$\text{free transport} \equiv -\mathbf{c} \cdot \frac{\partial}{\partial \mathbf{r}}(nf) \quad (1.4)$$

If an external force, F , acts on the system, its contribution to the change in particle distribution can also be expressed as a dot product (equation 1.5). In this case, the dot product of the external force and the change of the particle distribution with respect to velocity (the velocity class $\mathbf{c}$).

$$\text{external influences} \equiv -\mathbf{F} \cdot \frac{\partial}{\partial \mathbf{c}}(nf) \quad (1.5)$$

³K. Huang, Statistical Mechanics, (Wiley, New York, 1963), Chap. 3.

Finally, the distribution changes due to collisions occurring among the particles is calculated. The overall increase of particles of class $\mathbf{c}$, by the collision of particles of another class $\mathbf{c}_1$, with the particles of class $\mathbf{c}$ is expressed in integral form by two terms (equation 1.6). Given f^* is the post-collision distribution, the first term in the integrand represents the injection of particles into class $\mathbf{c}$, while the second represents the removal of particles from class $\mathbf{c}$.

$$\text{particle interactions} \equiv \int_{-\infty}^{\infty} \int_0^{4\pi} n^2 (f^* f_1^* - f f_1) (\mathbf{c} - \mathbf{c}_1) \sigma d\Omega d\mathbf{c}_1 \quad (1.6)$$

The complete Boltzmann equation can therefore be expressed as the sum of these three particle distribution changing processes (equation 1.7).

$$\frac{\partial}{\partial t}(nf) = -\mathbf{c} \cdot \frac{\partial}{\partial \mathbf{r}}(nf) - \mathbf{F} \cdot \frac{\partial}{\partial \mathbf{c}}(nf) + \int_{-\infty}^{\infty} \int_0^{4\pi} n^2 (f^* f_1^* - f f_1) \mathbf{c}_r \sigma d\Omega d\mathbf{c}_1 \quad (1.7)$$

Mean Free Path and Collision Frequency

Two common microscopic parameters used to characterize particle interactions, are the collision frequency and the mean free path. The *mean free path*, λ , is the average distance in which a particle travels between successive collisions. It is usually related to the *collision frequency*, ν , as

$$\lambda = \bar{c} / \nu, \quad (1.8)$$

where $\bar{c}$ is the mean particle speed.

The collision frequency is defined as the average number of collisions a particle undergoes per unit time. Although particle collisions can become quite complex, a

simple heuristic picture can be constructed by moving a test particle through a static particle field (Figure 1.3). Since every particle has some collision cross-section, σ , as the particle moves one time unit at velocity c_i , it sweeps out some amount of volume based on its geometry. Any particle found within this volume element can be considered a collision event. Therefore, the particle's collision frequency is the accumulated number of particles (per unit time) found within this volume.

Figure 1.3. A particle sweeping out a cylindrical volume as it transverses a static particle field.

In the case of a fixed cross-section, σ , (e.g., a hard sphere) the cylindrical volume swept by a particle moving at speed c_i , is $\sigma c_i \Delta t$, and the collision frequency is $\sigma n c_i$.

Knudsen Number

A convenient measure of a particle system and its relationship to the *characteristic dimension*, L , is its *Knudsen number*, Kn .

$$(Kn) = \lambda / L \quad (1.9)$$

This dimensionless number is frequently used as a valid criterion for continuum fluid models. Hydrodynamic equations, like Navier-Stokes, are only valid in the continuum

regime. For most hydrodynamic systems, the continuum approximation is valid if the Knudsen number is sufficiently small ($Kn \leq 0.1$).

It is possible to graph the continuum regimes as they relate to density and characteristic dimension. In Figure 1.4, the continuum limit for air under standard conditions ($d = 3.7 \times 10^{-10}$ m) along with the dilute gas threshold are shown.⁴ These limits reflect $(Kn) = 0.1$ for the continuum regime and $\delta/d = 7$ for a dilute gas.

Figure 1.4. The continuum regime for air as a function of density and characteristic length.

Hard Sphere Model

Although real molecules typically have their particle-particle forces characterized by some potential function, like the Lennard-Jones potential, it is possible to simplify both

⁴G. A. Bird, Molecular Gas Dynamics, (Clarendon Press, Oxford, 1976), p. 17-19.

mathematically and computationally any realistic potential function with an approximation, the simplest being the hard sphere model.

The hard sphere is a particle model which approximates the non-linear potentials of real molecules (i.e., the attraction and the repulsion forces found in particle-particle interactions) with a solid, purely elastic sphere of some effective diameter, d_{eff} . By making use of the effective diameter and the spherical symmetry of the particle's geometry, the collision cross-section is simply,

$$\sigma = \pi d_{eff}^2 . \quad (1.10)$$

Furthermore, the mean free path of the hard sphere can be shown to be,

$$\lambda_h = \frac{m}{\sqrt{2} \pi d_{eff}^2 \rho} \quad (1.11)$$

where m is the particle mass and ρ the density. Note that for hard spheres, the mean free path is independent of temperature.

Variable Hard Sphere Model

Although the simple hard sphere model can be useful in some molecular simulations, it does not accurately model a real gas (e.g., air). It was found for example, that the hard sphere model does not properly represent the dependencies that temperature and velocity have on viscosity. One way of improving the hard sphere model, and in doing so simulate real potentials (e.g., the Lennard-Jones potential), is by changing the collision cross-section of the particles. This can be accomplished by varying the effective diameter of a hard sphere and is the motivating concept behind Bird's variable hard sphere (VHS) model,

$$d_{eff} = d_{ref} \left[\left\{ 2kT_{ref} / (m_r c_r^2) \right\}^{\omega-1/2} / \Gamma(5/2 - \omega) \right]^{1/2} \quad (1.12)$$

where T_{ref} is a reference temperature, ω a free parameter, c_r the relative speed, k the Boltzmann constant, Γ the gamma function and m_r the reduced mass.⁵ The reference diameter d_{ref} , of the particle is given as,

$$d_{ref} = \left[\frac{15(mkT_{ref}/\pi)^{1/2}}{2(5-2\omega)(7-2\omega)\mu_{ref}} \right]^{1/2} \quad (1.13)$$

where μ_{ref} is some reference viscosity at temperature T_{ref} . The mean free path of a variable hard sphere can thus be expressed as,

$$\lambda_{vhs} = \frac{1}{2^{1/2} \pi d_{ref}^2 n} \left(\frac{T}{T_{ref}} \right)^{\omega - \frac{1}{2}}. \quad (1.14)$$

Note that the variable hard sphere model reduces to the hard sphere model if $\omega = 1/2$.

Viscosity and Reynolds Number

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

$$\text{momentum flux} = -\eta \nabla \mathbf{u}, \quad (1.15)$$

⁵G. Bird, in Microscopic Simulations of Complex Hydrodynamic Phenomena, Proceedings CECAM Conf., Sardinia, July 1992.

where $\mathbf{u}$ is the stream velocity. A related quantity is the *kinematic viscosity*, ν , which is defined as the dynamic viscosity over the density, $\nu = \eta/\rho$. As we shall later see, viscosity can be calculated in an equilibrium particle system by the measurement of the macroscopic stream velocities.

From a system perspective, viscosity can be thought of as the friction exerted between particles in order to bring them to equilibrium. As the system approaches equilibrium the average velocity of the particles will become uniform (but not necessarily homogeneous at the system level which is dependent on the thermal driving forces involved). Therefore, the dynamic viscosity of a system can also be thought of as the rate in which average velocity is transported.

Viscosity, however, is just one descriptive component of a fluid flow. A parameter used to qualitatively describe and compare fluid systems is the *Reynolds Number*, Re . The Reynolds number of a system relates the density, ρ , stream flow, u , characteristic length, L , and viscosity into a single dimensionless quantity,

$$Re = \frac{\rho u L}{\eta} = \frac{u L}{\nu}. \quad (1.16)$$

Mach Number

Another important parameter used to describe a fluid is the speed, a , in which transversal waves (e.g., sound) move through a fluid medium. The proportionality of the stream flow, u , with a is referred to as the flow's Mach number, Ma ,

$$Ma = \frac{u}{a}. \quad (1.17)$$

For an ideal monatomic gas, a is expressed as,

$$a = \sqrt{\gamma \frac{kT}{m}} \quad (1.18)$$

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

The Mach number of a flow is a dimensionless number useful in characterizing the relative densities and momentum transport of fluids. When the Mach number is low ($\text{Ma} < 0.3$) the flow is, to a good approximation, incompressible. At high Mach number ($\text{Ma} > 1$), discontinuous effects, like shock waves, appear; these phenomena can be difficult to handle computationally using continuum models.

Note that a similar quantity related to a , is $\bar{c}$, the average particle speed in an ideal gas due to thermal energy. In an equilibrium gas,

$$\bar{c} = \sqrt{\frac{8}{\pi} \frac{kT}{m}} = \sqrt{\frac{8}{\pi\gamma}} a. \quad (1.19)$$

Boundary Conditions

Although the previously discussed definitions and concepts focused on the dynamics of particle-particle interactions, and the quantitative parameters used to characterize them, a system's dynamics is often driven by boundary interactions. In fact, boundary conditions (i.e., types of interactions) bridge the gap between theoretical models, such as Kac's simple equilibrium cell, and physically realizable applications.

When a particle strikes a boundary (e.g., a wall), two primary interactions occur. First there is a change of momentum. If the collision is elastic, conservation of momentum is maintained as the particle and the boundary exchange momentum.

Simplifications usually assume a rigid boundary, that is, one with very high mass, and has purely elastic collisions. This causes the particle to simply reflect off the boundary's surface. A boundary that only affects momentum is referred to as *specular*.

The second type of particle-boundary interaction is thermal energy transference. When a particle strikes a boundary that has some amount of internal energy, the particle exchanges energy in the form of kinetic energy. In the case of inelastic collisions (a more realistic situation), both types of interactions occur and there is an exchange of both thermal energy and momentum between the boundary and the particle.

Typically in a computational simulation, some assumptions are made to simplify the boundary interfaces without sacrificing the overall physics of the system. For example, boundary walls may be held at a constant temperature; this is equivalent to the wall being a thermal reservoir. No amount of energy exchanged at the boundary will change the temperature of the wall. Since the boundary is in effect independent of the processes occurring around it, the boundary can be used in a simulation (model) as an “ideal” heat source or sink.

Thermalization

The process used to model a particle's interaction with a thermal boundary (a wall that has some amount of internal energy or temperature) and compute the energy exchanged is called *thermalization*. Since temperature itself is a macroscopic property, thermalization is usually approached as a statistical phenomena. That is, when a particle strikes a thermal boundary the energy gained is a function of temperature but is applied to the kinetic energy of the particle in a stochastic manner. In a three dimensional system, a particle will not only leave the boundary at some random speed but also in a random direction.

Figure 1.5. Thermalization of a particle colliding with a wall and the subsequent velocity distributions. The top graph shows a Maxwell-Boltzmann velocity distribution, while the bottom graph shows a biased Maxwell-Boltzmann velocity distribution.

Although the interactions appear purely random at the particle level, it can be shown that the overall velocity distribution of all the particles after they interact with the thermal boundary, follow a Maxwell-Boltzmann distribution (Figure 1.5). Of course, given a boundary that is planar and solid, the velocity of particles perpendicular to the plane after thermalization, must always be away from the boundary. Therefore, the velocity distribution perpendicular to the boundary plane is a biased Maxwell-Boltzmann distribution (see Figure 1.5).

Velocity Slip and Temperature Jump

Due to the relative scarcity and discreteness of particle interactions, a number of non-continuous phenomena will occur at close proximity (i.e., less than a mean free path length) to a pure thermal boundary. Two of the most common and observable of these effects are *velocity slip* and *temperature jump*.

Given a purely static thermal boundary (e.g., a non-moving wall) and some constant fluid flow driven by a uniform potential field (i.e., the flow is at equilibrium) parallel to the axis of the wall, velocity slip will occur.

Figure 1.6. Flow within one mean free path of a thermal boundary. Note that the dotted lines represent the linear extrapolation of the stream flow velocity with boundary intersections defining the velocity slip.

Since the boundary is static, one might expect that the velocity flow at the boundary to be zero. This is only strictly true at the macroscopic level. Using a microscopic scale, there is always an observable flow within one mean free path of the boundary. If we extrapolate the flow of the velocity stream close enough to the boundary interface so that linearity can be assumed, it becomes apparent that the stream velocity will not reach zero until well into the boundary (Figure 1.6).

There are two types of particles at the boundary, those heading away from the boundary (i.e., already interacted), and those approaching the boundary. The particles leaving the boundary are purely thermalized, and since thermalization randomly orients

their vectors, they have a net zero velocity along the plane of the wall. However, the stream flow particles which have had no interactions with the thermal wall (they have not reached it yet), have a net velocity along the plane of the wall. It is the contribution of these non-thermalized particles (the ones that have not interacted) that produces a velocity flow (i.e., velocity slip) at the thermal boundary.

A second observable effect due to thermal boundaries and also directly related to the velocity slip phenomenon previously discussed, is temperature jump. Temperature is a macroscopic property computed from the kinetic energy of an ensemble of neighboring particles (assuming ideal gas type particles $KE = 1/2mv^2$). Close to the thermal boundary (i.e., within one mean free path) insufficient particle-particle interactions (i.e., collisions) occur between the thermalized particles and the streaming particles to produce a local equilibrium (i.e., there are not enough collisions to disperse the thermal energy). Therefore, the temperature close to the boundary is not equal to the boundary's temperature which causes a temperature discontinuity to be observed. Only when the temperature gradient is zero does a local equilibrium develop and the temperature gradient between the boundary and the boundary surface become continuous.

Simple Flows

There are many types of simple particle systems that can be used to study viscosity. Due primarily to the complementary methods in which system viscosity can be calculated, Poiseuille flow and Rayleigh-Bénard flow provide excellent simulation choices. In the Poiseuille flow system, viscosity manifests itself as a transport phenomena (stream velocity) while in the Rayleigh-Bénard system, viscosity is coupled to convection phase transitions.

Poiseuille Flow

Poiseuille flow, or planar pipe flow as it is sometimes called, consists of fluid flowing between two parallel walls in a two dimensional system (Figure 1.7). Driving

the flow is some force that exists parallel to the walls. Typically, the wall boundaries are thermal (i.e., reservoirs) while the boundaries at the ends of the walls are periodic (i.e., a particle exiting at one end of the system will reappear at the other end). Boundary periodicity is used to simulate infinite wall length (infinite boundaries are mathematically simpler to handle from a differential equation perspective) while maintaining computational feasibility.

Figure 1.7. A planar Poiseuille system with thermal walls at temperature T , the width of the system, L , and a uniform gravitational force, g . Note that parabolic shape of the flow vectors.

Given realistic fluid properties, a uniform potential field parallel to the walls, and realizable thermal forces (i.e., temperature), a Poiseuille system quickly reaches steady state. The flow of the particles is predominantly along the axis parallel to the walls (i.e., the stream flow), while the flows perpendicular to the walls approach zero (microscopic interactions in these directions are random).

Figure 1.8. The velocity profile of a typical Poiseuille cross-section (i.e., a swath). A second order polynomial fit was calculated using velocity samples taken across the stream.

By looking at a cross section (swath) perpendicular to the stream flow, a one dimensional velocity profile can be calculated. Excluding the particles in close proximity to the walls (within one mean free path, the particles still reflect the thermalization of the walls), the velocity distribution is parabolic.

It is possible to obtain the steady state velocity profile (swath across the stream), $v(x)$ by solving,

$$\eta \frac{d^2 v}{dx^2} = \rho g \quad (1.20)$$

using the boundaries $v(\pm L/2) = v_0$, where v_0 is the velocity slip, L the distance between walls, g the gravitational acceleration, ρ the density and η the dynamic viscosity.⁶ The solution,

$$v(x) = \frac{\rho g}{2\eta} x^2 + \left(v_0 - \frac{\rho g L^2}{8\eta} \right) \quad (1.21)$$

is a second order polynomial (parabolic curve). Therefore, it is possible to determine the viscosity of a system from the curvature of the velocity profile.

Rayleigh-Bénard Flow

A second type of particle system that can be used to calculate viscosity is called Rayleigh-Bénard flow. This system is physically characterized by two thermal parallel plates (simply walls in two dimensions) which maintain some finite temperature difference and a uniform potential field perpendicular to the axis of the plates. There is always a balance maintained between the opposing forces. That is, the amount of potential energy available in the potential field (e.g. gravity) is equivalent to the thermal potential between the plates. When the temperature difference between the two plates is very small, thermal conduction dominates and there is no net flow in the fluid. If the temperature difference between the two plates exceeds some threshold value, a convection phase transition occurs. In fact there are a number of convection phase transitions that can occur as the temperature is increased. These transitions manifest themselves as increasing orders of convection flows (e.g., a double flow). It is the quantification of these threshold temperature differences that can be coupled to the viscosity of the system.

⁶L. D. Landau and E. M. Lifshitz, Fluid Mechanics, (Pergamm Press, Oxford, 1959), Chap. 2.

Figure 1.9. A two-dimensional Rayleigh-Bénard system with two thermal boundaries, ($T_1 > T_2$), two specular boundaries (perpendicular to the thermal walls), a system height and width of L , and a uniform gravitational force, g .

Although, Rayleigh-Bénard flow can be modeled with periodic boundaries (i.e. infinite length) connecting the thermal walls (plates), purely specular boundaries can also be used. While periodic boundaries must be physically separated by a large distance in order to allow two counter rotating rolls to develop, specular boundaries only require enough distance for one convection roll. This promotes smaller physical system dimensions when specular boundaries are used, and reduces the total number of particles in the system.

Rayleigh Number

The dimensionless number that relates the parameters needed to induce convection (phase transition) in a Rayleigh-Bénard system, is called the *Rayleigh number*,⁷ Ra , and is expressed as,

$$Ra = \frac{\alpha g L^4}{\nu \kappa} \quad (1.22)$$

where g is the gravitational field, L the depth of the system, κ the thermometric conductivity, ν the kinematic viscosity, $\gamma = \Delta T / \Delta x$ is the uniform temperature gradient, and $\alpha = 1/\rho [\partial \rho / \partial T]_p$ the coefficient of volume expansivity.

The point (i.e., Rayleigh number) in which the system starts convection is called the *critical Rayleigh number*, Ra_c . By making use of the Boussinesq approximation, the critical Rayleigh number (stability criteria) for a given system can be calculated from the Navier-Stokes equations. The Boussinesq approximation assumes that the density of the system is constant except in the gravitational acceleration term. Since this implies that density change in the system is only dependent on temperature change (i.e., negatively proportional to), the Navier-Stokes equations can be simplified in terms of temperature and velocity. By looking at the first order perturbation of these equations, the stability of the system can be determined. If the perturbations damp out, then the system is in a pure conduction state. However, if the perturbations grow exponentially, then the system is unstable and convection starts. The critical Rayleigh number is thus the critical value (point) in which an exchange of stability occurs between a conduction solution and a convection solution.

⁷S. Chandrasekhar, Hydrodynamic and Hydromagnetic Stability, (Dover, New York, 1961), Chap. 2.

CHAPTER 2

COMPUTATIONAL METHODS

Computational fluid flow models take two basic forms, direct equations for a macroscopic perspective, and particle simulation for a microscopic view. In the former, a series of partial differential equations (PDE) like the Navier-Stokes equations, are solved directly (macroscopic solutions).¹ Since only the simplest boundary conditions for these equations can be expressed in closed form, computational algorithms are used to generate the solutions (approximations). In general, all of the modeling (i.e., work) is intrinsic to the equation set up and the applications of the boundary conditions. And because the computational algorithms are specific to the physical aspects of the model, the fluid flow problem becomes a mathematical exercise in differential equation solving.

The second and more recent approach to fluid flow modeling is direct particle simulation. In these methods, which include Molecular Dynamics, fluid flow is modeled and simulated at the particle or microscopic level. Every particle interaction and boundary condition is modeled using deterministic Newtonian dynamics (i.e., first principles). This set of ordinary differential equations (ODE) can then be solved directly by the repeated application of simple computational algorithms like Runge-Kutta or Verlet.

Direct Equations

Navier-Stokes

A number of hydrodynamic equations can be used to express fluid flow. The best choice for a given system depends on the boundary conditions and the physical parameters used. Each set of hydrodynamic equations relates the same fundamental laws

¹D. A. Anderson, J. C. Tannehill and R. H. Pletcher, Computational Fluid Mechanics and Heat Transfer, (Hemisphere, New York ,1984).

of physics—conservation of mass, conservation of energy and momentum (Newton's second law)—as the macroscopic properties of density, temperature and flow.

One of the best known hydrodynamic formulations, the Navier-Stokes equations, express continuity (conservation of mass), momentum (forces acting on a system) and thermal transport (conservation of energy) in differential form. The continuity equation is expressed as,

$$\frac{\partial \rho}{\partial t} + \nabla \cdot \rho \mathbf{V} = 0; \quad (2.1)$$

the momentum equation as,

$$\frac{\partial \mathbf{V}}{\partial t} = -(\mathbf{V} \cdot \nabla) \mathbf{V} - \frac{1}{\rho} \nabla P + \nu \nabla^2 \mathbf{V} + \mathbf{g} \quad (2.2)$$

and the energy equation,

$$\frac{\partial T}{\partial t} = -(\mathbf{V} \cdot \nabla) T - \frac{P}{\rho c_v} \nabla \cdot \mathbf{V} + \frac{\kappa}{\rho c_v} \nabla^2 T \quad (2.3)$$

where ρ is the density, $\mathbf{V}$ the fluid velocity, $\mathbf{g}$ is the gravitational acceleration, P the pressure, ν the kinematic viscosity, c_v the specific heat and κ the thermal diffusivity.

There are two common methods of solving hydrodynamic equations, relaxation and marching. Both of these methods are themselves general algorithms and are commonly used to solve many classes of problems involving differential equations and boundary conditions.

Relaxation Method

In a relaxation application, the fluid flow model is defined by a set of partial differential equations, various boundary conditions, and a set of initial parameters. Unfortunately, the relaxation method only allows for the computation of steady state (i.e., time invariant) flows. However, by assuming steady state, the time dependent derivatives can be set to zero thus simplifying the hydrodynamic equations. If a problem contains sufficient boundary complexity, a series of equations must be used to separate the problem into various boundary regions.

The typical solution methodology for a set of Navier-Stokes equations takes these general steps:

1. The Navier-Stokes equations are rewritten with some simplifying assumptions (e.g., time invariant flow and constant temperature).
2. The equations are then expanded into scalar form (in order to fully represent the dimensionality required for the problem) and restructured into difference equation form (i.e., one, two, or three dimensional flow). Also, these coupled equations can be rewritten with variable substitutions that represent hydrodynamic properties such as vorticity and stream function.
3. The simulation space is then divided into a series of uniform lattice spaces or volume cells. Note that for convenience, boundary conditions are usually placed on the lattice edges. Also, algorithms may be written to handle various boundaries anomalies specific to the problem (e.g., thermal vs. specular boundaries).
4. Finally the relaxation process is initiated by choosing trial values which correspond to a simple boundary solution. By applying these values at the lattice points, an approximate solution is returned. These values are then used to compute a better solution, and fed back into the difference equations. The process of feeding approximate solutions back into the discretized hydrodynamic equations is iterated

until the system converges to a computational equilibrium. Note that the rate of convergence can be controlled by limiting the amount each approximate solution influences the next iterative set of input values.

Since the relaxation method requires extensive “up front” work defining boundaries and the equation parameters, most simulations become problem specific. For example, a simulation designed to model Poiseuille flow will not be able to simulate Rayleigh-Bénard flow without redefining the boundaries. When a boundary condition is modified it usually becomes advantageous to repeat the design of the whole model (i.e., a new algorithm).

Marching Method

Another method commonly used to solve hydrodynamic PDEs is called the marching method. Unlike the relaxation method, which excludes time derivatives and only returns a steady state solution, the marching method allows for a slow progression (or march) in time to be modeled. At each time step or state, the marching method returns the complete fluid flow solution. By contrast, only the final state of the relaxation method is considered a valid solution. The computed intermediate relaxation states are at best only approximate, and are useful for numerical convergence purposes only.

Marching methods convert a simple but coupled set of hydrodynamic partial differential equations,

$$\frac{\partial S}{\partial t} = G(S(\mathbf{r}, t)) \quad (2.4)$$

into difference equation form,

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

$$S(r, t + \Delta t) = S(r, t) + \Delta t G(S(r, t)) , \quad (2.5)$$

New Old Change

where S , some state property (e.g., temperature or density), is a function of time, t , and space, r , and G is a function relating other dependent parameters including past states.

Simply stated, the next state of a system is the sum of the old state plus the changed state at some Δt time step. Internal to the G function are any derivative functions or equations that reflect spatial relationships (e.g., gradients).

A Marching method fluid flow model can be implemented as follows:

1. A hydrodynamic equation is converted into difference form. This includes the spatial relation function G and any boundary conditions.
2. The system space is quantized into a uniform number of lattice points, and the initial density state for those points at $t = 0$, is defined. These points are analogous to sampling points in a real fluid system.
3. The next system state at time $t + \Delta t$, is determined by directly solving the difference equations. The function G , is solved by sweeping through all lattice points and computing any required derivative values from the current states of the spatial neighbors (sample points). Note that boundary conditions are expressed as known lattice state values.

Problems With Direct Equations

The coupled, non-linear partial differential equations that are used to express hydrodynamic fluid flow, can be quite complicated to solve. Usually, only a few simple boundary conditions, which facilitate equation simplification, can readily be solved. The primary advantage of these equations is that they are general, and therefore can be

applied to a number of problems. Generality though does not necessarily imply simplicity. Whenever boundary conditions in a fluid system are changed, it may become expedient and even necessary to redesign the computational algorithm used to solve the hydrodynamic equations. However, even when a computational algorithm is appropriate, most solutions are limited by numerical stability.

Typically, the hydrodynamic equations are used to model the state flow properties of a fluid and extract macroscopic properties such as temperature, viscosity, and vorticity. Unfortunately, this type of hydrodynamic model will only remain valid as long as the system satisfies the continuum approximation. The continuum approximation states that in order for a system's macroscopic properties to be considered continuous, the effects of individual molecule interactions on system dynamics must be negligible. That is, the discrete nature of the fluid (e.g., particles) is sufficiently resolute to be considered continuous (from a macroscopic perspective). From a statistical perspective this means that a continuum approach is only valid if there are a sufficient number of particles present in some volume element of the fluid to compute useful averages. This threshold limit is usually characterized by a Knudsen number around one (refer to Figure 1.4). At this point, the hydrodynamic equations start to break down and significant variations occur between the predicted model (e.g., Navier-Stokes) and real fluid flow.

Direct Particle Simulation

Molecular Dynamics

One of the first direct particle simulation methods, Molecular Dynamics, was developed by Alder and Wainwright in 1957.² Molecular Dynamics is a general algorithm which models the qualitative properties of a dynamic fluid system through the

²B. J. Alder and T. E. Wainwright, "Phase transition for a hard sphere system," J. Chem. Phys. 27, 1208 (1957).

incremental (in time) computation of individual particle trajectories and interactions.³ Unlike the relaxation and marching methods, which setup and solve hydrodynamic equations directly, Molecular Dynamics uses simple Newtonian formulations (ODEs) and particle potentials to model the microscopic interactions and simulate the macroscopic fluid properties.

Fig. 2.1. A two-dimensional particle system partitioned into equally spaced cells. The vector associated with each particle represents the instantaneous velocity.

Molecular Dynamics models a fluid system by mapping a set of “useful” physical characteristics and boundary conditions onto a simulation space (i.e., some coordinate scheme). Within this simulation space the particles are tracked as they move and interact with each other. The algorithm itself is responsible for moving the particles using

³M. P. Allen and D. J. Tildedley, Computer Simulation of Liquids, (Clarendon, Oxford, 1987), Chap. 3.

Newtonian dynamics, and determining any particle-particle and particle-boundary interactions. By partitioning the simulation space into equally spaced cells or lattice points (Figure 2.1), the macroscopic fluid properties of the system can be extracted via statistics.

The basic Molecular Dynamic algorithm is composed of just a few steps:

1. Divide the simulation space into uniform cells or lattice points (Figure 2.1) and populate the cells with some initial distribution of particles.
2. Move all particles in the simulation space one time step using Newtonian dynamics and store the position and velocity values. Since the Newtonian equations can be expressed as simple first order ODEs, a Runge-Kutta type numerical algorithm is computationally quick and simple.
3. Check the proximity of each particle to all other particles and determine the inter-particle interactions. For all the identified proximate particles, recalculate their forces using a particle-potential function (e.g., the Leonard-Jones potential) and determine their corresponding trajectories .
4. Recalculate the positions and velocities of any particle crossing a boundary (e.g., a thermalizing wall, periodic boundary, specular wall).

Proper time step values are particularly important in the Molecular Dynamics model (as well as most particle simulation techniques). Since the value of the time step is integral to the calculation of future particle positions (i.e., solving the simple ODEs), the particle-particle interactions (e.g., potentials) are directly correlated to the distance each particle moves per time step. That is, it is possible to violate energy conservation if the time step used is not small enough to handle the fastest particle in the system. If the simulation time step is too large, colliding particles with high relative velocities may find themselves on top of each other (i.e., unusually close proximity), within one time step.

Furthermore, if a realistic particle-particle potential like Lennard-Jones is used, then the close proximity of the particles will cause them to gain an unrealistic amount of kinetic energy.

Direct Simulation Monte Carlo

Although Direct Simulation Monte-Carlo (DSMC) is a direct descendent of the Molecular Dynamics approach to particle modeling, the algorithm takes advantage of the binary collision property of dilute gases, and in doing so takes a stochastic twist. Originally proposed by Bird in 1961, DSMC moves particles in the typical Molecular Dynamics fashion (i.e., Newtonian dynamics).⁴ However, binary particle-particle collisions, which predominate a dilute gas, are chosen randomly. The basic premise behind Bird's stochastic particle interactions was developed in Kac's simple dilute equilibrium gas model (1959).

Kac mathematically showed that a given dilute gas, existing in some arbitrary but homogeneous distribution, will have its particle ensemble velocity distribution converge to the Maxwell-Boltzmann distribution as it approaches equilibrium. Given constant state properties (i.e., temperature, pressure, volume, etc.), the velocity distribution that develops as equilibrium is approached, is a direct result of particle-particle interactions (i.e., collision). Interpreting these ensemble velocity distributions as probabilities, a direct correlation can be made between any two particle velocities (in a given system) and the likelihood of their collision.

Using the velocity distribution of a dilute gas as a probability property (distribution) of that gas, Bird was able to stochastically pick particle pairs in a system and determine the likelihood of a collision based on the relative velocity between the pair. Therefore, given a relatively homogeneous fluid density and the velocity profile of some space (e.g.,

⁴G. A. Bird, "Approach to translational equilibrium in a rigid sphere gas, " Phys. Fluids, 6, 1518 (1963).

a cell), the number of collisions that will occur in some delta time step, can be calculated. In Molecular Dynamics, it is necessary to track each particle and determine every interaction. But in Bird's algorithm (DSMC), the associated probability of interaction between any chosen particle pair is dependent on its relative velocity. Once a collision pair is chosen, the collision is calculated independent of the pair's location. That is, although the pair may not be spatially proximate, the collisions are effectively calculated as if they were (as if the particles were right next to each other). Note that the colliding particles are not physically moved during the collision. However, the calculated exit velocities are applied to the collision pair. At the next time step, the free flight (i.e., the movement of the particles) is then calculated using the post-collision velocities.

The typical DSMC algorithm is composed of just a few simple steps:

1. Divide the simulation space into uniform cells or lattice points (Figure 2.1) and populate the cells with some initial distribution of particles.
2. Move all particles in the simulation space one time step using Newtonian dynamics and store the position and velocity values. Since the Newtonian equations can be expressed as simple first order ODEs, the use of a Runge-Kutta type numerical algorithm is computationally quick and simple.
3. Determine the number of particle collisions that could occur at each time step in each cell. Then randomly pick two particles in a given cell, determine the relative velocity between the particles, and check the probability of their collision.
4. If the particles collide, determine their new velocities without changing their positions.
5. Recalculate the positions and velocities of any particle crossing a boundary (e.g., a thermalizing wall, a periodic boundary, or a specular wall).

Problems With Particle Simulations

Unlike the macroscopic or direct equation hydrodynamic models, the particle or microscopic models are not limited to the continuum regime. However, where the solutions to macroscopic equations are absolute (within the continuum regime) the very discrete nature of particle models causes them to suffer from statistical fluctuations. This underlies a fundamental difference between the two approaches. The macroscopic models use the macroscopic properties to estimate or derive the microscopic interactions within a fluid. Conversely, the microscopic models use the statistical averages of many individual particles to deduce the macroscopic properties of the fluid. This is basically a top down versus bottom up approach, and the method chosen for a particular system model is directly coupled to the spatial scale required.

Ironically, the higher density continuum regions which define the limits of a valid macroscopic approach, tend to constrain many microscopic simulations in both size and versatility. In order to properly extract macroscopic properties, individual particle interactions must be tracked. This means that the position and velocity vectors of every particle must be calculated and stored at each time step. But the size of the system (i.e., the number of particles as well as the physical space) is limited by both storage space (i.e., physical as well as virtual memory) and computational speed (i.e., raw computer speed or algorithm efficiency). And since generally the size of the modeled system is correlated to the density of the fluid or the scale that the system views (e.g., 1 meter versus 1 micron), it may be virtually impossible to simulate certain realistic problems. For example, for air (which is typically considered dilute) one mole (6.02×10^{23} molecules) at standard operating conditions (273K), with today's computers, using Molecular Dynamics will take an exorbitant amount of time and resources.

Another particle simulation problem occurs outside the continuum regime. At Knudsen numbers over one, the scarcity of particles in the system induces statistical

fluctuations in the computed flow. In order to preserve the particle approach when statistical fluctuations predominate, it is possible to use ensemble or time averaging. By taking the ensemble or time average of a system, a probabilistic description of the fluid flow is accumulated and statistical fluctuations are reduced. This process extends the particle model into sparser density regimes which are inaccessible to any macroscopic approach.

However, there is a limit to this averaging process. The ratio of the characteristic length of the system to the mean molecular space must in general be greater than 100 (i.e., $L/\delta > 100$). After this threshold point, the sample within the volume element is either too small for ensemble averaging or too dynamic for time averages (i.e., too many particles will cross a volume element border during the averaging time sample) and statistical fluctuations again dominate the model. Thus, in the extremely low density case of dilute gases, there may be insufficient particles present at a particular spatial scale to allow for proper flow resolution (i.e., macroscopic properties).

Note that although the DSMC (Bird) method is a derivative of the Molecular Dynamics approach, it allows a small number of particles to represent many physical particles in order to reduce the overall system size. Dilute gas systems can therefore be simulated using realistic time and memory resources. Unfortunately, the representation of a large number of particles by only a few number of simulated particles can cause significant statistical scattering. If we assume that this scatter follows the Poisson distribution and note that the standard deviation is proportional to the inverse square root of the number samples, then by time or ensemble averaging the correct particle flow will emerge out of the noise.

Method Comparison For Dilute Gases

Given the two basic computational approaches to modeling hydrodynamic fluids—the macroscopic or direct equation methods, and the microscopic or particle simulation methods—it is possible to determine the “best” approach in dilute gas modeling.

Macroscopic Methods

Macroscopic models only operate within the confines of the continuum regime. This limitation precludes the modeling of small scale systems and rarefied gases. That is, once either the scale or the density of the fluid system drops below the continuum regime (i.e., a Knudsen number greater than or equal to 0.1) the hydrodynamic equations breakdown and the model is invalid.

Therefore, the macroscopic approach is not suited to modeling the dynamics of a very dilute gas or very small scales ($L \approx 10\lambda$).

Microscopic Methods

The two particle algorithms discussed were Molecular Dynamics and Direct Simulation Monte Carlo. In both cases, the particle simulation approach will work quite well on dilute gases. The Molecular Dynamic approach is far more general in nature since all fluids, regardless of their density, can be modeled. On the other hand, the DSMC approach is limited purely to the dilute gas regimes by the validity of Boltzmann statistics which necessitate binary particle collisions.

The choice between Molecular Dynamics and DSMC thus becomes a purely pragmatic analysis of computational resources. Given that both methods have the same number of particles, the DSMC algorithm spends less time computing collisions due to the probabilistic approach (i.e., particle collisions are more easily identified and computed). Coupled with the ability of DSMC to represent large groups of particles by a single modeled particle, the effort needed to simulate a realistic fluid system, in both time and memory, is dramatically reduced.

Therefore, since computational resources by definition are finite, the “best” computational method for modeling dilute gases, is Direct Simulation Monte Carlo.

CHAPTER 3

DSMC SIMULATIONS

In order to study the effects that biased collision rules (i.e., preferred exit velocities) have on the viscosity of a particle system, a Direct Simulation Monte Carlo (particle simulation) dilute gas model was developed. The Direct Simulation Monte Carlo (DSMC) model was run in two general flow configurations, Poiseuille and Rayleigh-Bénard. As was previously discussed, these two simple flows provide complementary viscosity measurements. Viscosity is correlated to the velocity streams in Poiseuille flow while in Rayleigh-Bénard, viscosity is coupled to convection phase transitions.

The developed simulation was two dimensional and written in C (the programming language). The general algorithm, however, was based on Bird's one dimensional FORTRAN version originally published in 1961 and the updated improvements he outlined in 1991. Since the choice of C as the simulation's programming language was a matter of convenience (for the programmer), the code was intentionally written to follow the FORTRAN algorithm as closely as possible. This made direct comparisons possible between Bird's FORTRAN version and the new C code for one dimensional systems. However, with the addition of new collision rules, this work diverged from Bird's original algorithm.

Computational Resources

The code was originally developed, written and tested on a SUN SPARCstation using a few thousand particles and simple flow configurations (one dimensional) like Couette and simple thermal conduction. However, to study a realistic dilute gas system, hundreds of thousands of particles must be simulated. In DSMC, particle numbers are linearly correlated to memory usage (RAM) and computational calculations. Unfortunately, the SUN SPARCstations were found to be inadequate in both memory and

speed, and thus the simulations were run on the faster IBM RS/6000s at Lawrence Livermore National Laboratory (LLNL).

Like Bird's original FORTRAN code, the simulation code used in this work consists of three sequentially run sub-programs (a complete listing of the simulation code, along with C include files and comments can be found in the Appendix). All sub-program input and output is in the form of files. This allows for partial run storage and snapshot views of the system. For example, the state of the system (the flow) can be saved at any time and used as the initial state (i.e., conditions) in a later simulation run.

Figure 3.1. Flow diagram of the three sub-program implementation of DSMC.

Each of the three sub-programs handles a different portion of the simulation. The first program, called VCI2, initializes relevant simulation parameters and outputs the initial state of the system. The second program, VCR2, takes the state of the system

(input file) and moves the particles, calculates any interactions and provides statistical sampling capabilities. During this stage of the simulation, two output files are created. The first file stores the current state of the system (all the positions and velocities of the simulation's particles) while the second file contains the time averaged flow samples. The third and final program, VCO2, is used to calculate and analyze system properties such as viscosity and temperature from the sampled flow statistics.

Simulation Structure

A possible source of confusion when describing DSMC is the dimensionality of the simulation. Unlike early Molecular Dynamics models which used non-physical fluid dimensions (i.e., one or two), the dimensionality of a DSMC simulation refers to the geometry of the model and not the physical dimensionality of the fluid. That is, each DSMC cell always represents a three-dimensional volume of real space independent of the dimensionality of the simulation. For example, given a two-dimensional DSMC system, it appears planar, but actually has a thickness equal to one cell. Further, each DSMC particle interacts three-dimensionally in real space but only needs to be tracked in the simulation dimensions (i.e., the geometry of the DSMC simulation). This is possible because a particle's position is not used when evaluating collisions. Therefore, in a two-dimensional simulation, all three velocity components must be calculated (for collisions purposes), but only the x and y coordinates of each particle's position on the simulation plane need to be maintained in order to locate which cell the particle is in.

Although many simple flows like Couette, Poiseuille and simple thermal conduction can be adequately modeled in one dimension, more complex systems require greater degrees of freedom (i.e., more than one position coordinate). The convection flows seen in Rayleigh-Bénard for example, can only be observed in two (planar) or more dimensions. Therefore, a two-dimensional simulation system (geometry) was chosen to

provide a realistic modeling environment for both the Poiseuille and Rayleigh-Bénard simulations.

Unlike many simulation models which use dimension-less units, DSMC typically uses real physical values. For example, the simulation space is measured in meters and actual gas values like particle mass and cross-section are taken from experimental measurements. Therefore, subsequent comparisons of DSMC models can be applied directly against real systems or each other without unit ambiguities.

Table 3.1. DSMC parameters used in dilute gas simulations.

Dimensions	2
Number Cells	10,000
Mean Free Path (λ)	0.03174 m
Cell Width	0.47λ
Particle Type	Helium
Particle Mass	6.645×10^{-27} kg
Reference Viscosity (μ_{ref})	1.96×10^{-5} N s/m ²
Reference Temperature (T_{ref})	293 K
Number Density	1.4×10^{20} particles/m ³
Number of Simulation Particles	200,000
Averaged Sample	2,000 time steps
Average Wall Temperature	409.5 K

Although a table of the simulation parameters is provided for reference purposes (Table 3.1), there are many heuristics and calculations used to set up a workable simulation. The methodology used to arrive at these values highlights the interrelationships, assumptions, and problems involved in all DSMC simulations.

General Parameters

The particle used in all of the simulations discussed in this work was helium. There are a number of features which make the helium molecule an attractive particle choice. First, helium is a noble gas and thus inert (non-reactive). This means that the effects of chemical reactions in helium simulations can be neglected. Second, helium is a simple atom and its 's' electron shell is full. Since the 's' shell is spherical, it is plausible for a first order (i.e., simple) particle model to be a hard sphere (or similar type). Note that only hydrogen is simpler, but it tends to form covalent bonds (each atom shares an electron with its pair) and exists in diatomic form at standard temperatures and pressures. And third, helium exists as a gas under normal conditions. Since its liquefying point is low (4.21K), the range of temperatures applicable in a gas simulation are quite wide.

The average temperature in which a simulation operates is interrelated to many DSMC parameters. In the two flows studied (Poiseuille and Rayleigh-Bénard), thermal energy entered the systems exclusively through the constant temperature boundary walls (i.e., reservoirs). Comparison between data runs can only be made if the average energy entering the system is the same. Therefore, in every simulation run the mean of all boundary temperatures was 409.5K. While the Poiseuille flow had both thermal walls held at 409.5K, in Rayleigh-Bénard, a temperature gradient was applied to the system and the mean temperature between the upper and lower thermal walls was held at 409.5K (independent of temperature gradient). It is particularly important in a Rayleigh-Bénard system that the average boundary temperature allows for the proper simulation of the highest temperature gradient required. If the temperature of a boundary wall becomes too

low, the gas approaches liquid temperature phases and the DSMC simulation becomes unrealistic (not a dilute gas). Therefore, the value of 409.5K was arbitrarily chosen to allow for the proper simulation the largest temperature gradient applied, 546K.

The average boundary temperature provides a good estimate of the system's temperature at equilibrium (excluding potential fields like gravity which also add significant energy). Given this average temperature a number of estimates can be determined including average particle speed and the collision frequency. For example, in this work, using the average particle speed equation for an ideal gas (equation 1.19), and the average temperature, 409.5K, the average molecular speed, $\bar{c}$, for helium is roughly 1,470 m/s. Furthermore, the collision frequency, ν , is dependent on the molecular speed and the mean free path by $\nu = \bar{c}/\lambda$. Using Bird's variable hard sphere model (equations 1.12, 1.13, and 1.14), a rarefied density of 1.4×10^{20} particles/m³ (at standard temperature and pressure, the density is 2.67×10^{25} particles/m³), a reference viscosity of 1.96×10^{-5} N s/m² at 293K, the mean free path is calculated to be 0.03174 m. Therefore, in this work, the number of collisions per unit time per unit volume of helium (i.e., $1/2 n \nu$) at 409.5K is 3.24×10^{24} m⁻³ s⁻¹.

In a DSMC simulation, cell width should generally be less than one mean free path to insure local homogeneity. In this work, the cell width was arbitrarily assigned as 0.473λ . This divides the simulation space into 1.0×10^4 cells, 100 cells in each dimension (47.3λ) with a distance of 1.5 m between the thermal walls. Therefore, the overall Knudsen number of the system, defined as the mean free path divided by the characteristic distance (distance between the walls), is 0.0211. This puts us marginally within the continuum region and well within the range of a particle model like DSMC.

Furthermore, given the average particle speed, the maximum time step value that can be used in the simulation (to move the particles and determine the collisions) can be determined. When the time step is too large, a kind of spatial collision alias can occur on

the order of the cell width. In reality, the faster a particle moves, the more likely it is to have a collision within intervening cells. However, in a simulation, if a particle travels more than the width of a cell in a single time step, then it effectively bypasses any interactions with intervening particles, and skews the system's flow dynamics.

The maximum time step $t_{\max}$, can be expressed as,

$$t_{\max} = \frac{L_c}{\bar{c}} \quad (3.1)$$

where L_c is the cell dimension and $\bar{c}$ the average particle speed. Given a cell dimension of 0.47λ and the average particle speed of 1470 m/s, in this work, $t_{\max}$ is 1.0×10^{-5} s.

Typically, the time step is significantly lower than $t_{\max}$ in order to allow each particle a couple of time steps before they transverse the cell's width. For example, in the Poiseuille simulations the time step was 9.4×10^{-7} s, while in the Rayleigh-Bénard simulations the time step was 9.4×10^{-6} s. Note that in the case of Poiseuille, the flow speed was on the order of the average thermal particle speed, the speed of individual particles moving with the flow was potentially doubled. Therefore, the time step had to be lowered considerably.

The number of particles in all the simulation systems, defined as the number density, was 1.4×10^{20} particles/m³. Using this number density in the dilute gas equation (equation 1.1), the molecular spacing is 1.93×10^{-7} m. However, to determine if the gas is dilute, the effective diameter of helium is also needed. If we use Bird's variable hard sphere model (equation 1.12), a reference temperature of 293K, and a reference diameter for helium of 2.32×10^{-10} m, the effective diameter used in the simulations is on the order of 2.0×10^{-10} m. Therefore, the molecular spacing is much greater than the effective diameter and the dilute gas equation is satisfied.

Given square simulation cells, the volume of the system is 0.3375 m^3 and the total number of particles in the system is 4.725×10^{19} . Unfortunately, simulating 4.725×10^{19} individual helium molecules is not practical even given LLNL's computer resources, therefore a particle representation scheme was applied. That is, each of the 2.0×10^5 simulated particles was chosen to represent 1.575×10^{15} real helium molecules. This creates an initial density distribution of 20 simulated particles per cell. The tradeoffs involved in determining the number of simulation particles are straightforward. The more particles simulated per cell the more calculations (movement and collisions). On the other hand, if the distribution of particles per cell is too small, than the simulation becomes unrealistic. With a small number of particles in a cell, a significant number of them will collide multiple times with the same particle before their momentum carries them across a cell boundary. This tends to promote the development of local equilibrium at the cell level. In order to avoid this, the number of collisions per cell per time step should be less than half the number of simulated particles in the cell. In other words, each particle should not collide more than once per time step.

Particle Collision Calculations

Central to the DSMC viscosity simulations is the calculation of the particle-particle interactions (i.e., collisions). Collisions are calculated every time step for each cell. The probability of a particle colliding with another particle is directly correlated to the relative velocity and density of all the other particles in the cell (see Kac's model) and follows a Maxwell-Boltzmann distribution. Calculation of individual particle probabilities at every time step is computationally intensive. Therefore, DSMC calculates collision probabilities on the cell level to reduce the overall particle calculations. The DSMC process takes two steps. First, calculate the maximum number of collisions that can

occur in each cell per time step, and then second, randomly select (with an acceptance criteria filter) the particles in each of those cells that do collide.

The first step, the determination of the maximum number of collisions, is calculated using an upper bound on the product of the relative velocity, c_r , (of the collision pair) and the cross-section, σ , (of the particles). That is, in any given time interval, the number of particle collisions that occurs increases in proportion to the volume swept by the particles. This velocity cross-section product is referred to as the maximum collision product, $C_{\max} = (c_r \sigma)_{\max}$, and is system dependent (e.g., in a Couette flow the wall velocity effects must be taken into account), and calculated during the simulation's initialization.

Succinctly stated, the maximum number of collisions that can occur in one time step in one cell is,

$$\text{maximum number collisions} \equiv \frac{1}{2} n N C_{\max} \Delta t \quad (3.2)$$

where n is the number density in the real gas, N the number of simulated particles in the cell, and $C_{\max}$ is the maximum collision product. Note that this number (maximum number collisions) is not the number of collisions that will occur, but the number of particle collision selections that will be attempted.

Once the maximum number of collisions in the cell for a particular time step is calculated, the second part of the DSMC process, particle pair selection, occurs. For each potential collision, two particles are randomly selected from that cell. In the rare case that fewer than two particles are present in a given cell, a nearest neighbor approach is implemented. That is, a particle collision partner is extracted from an adjoining cell (the probability of the adjacent cell providing the collision particle is based on the relative

density of particles in that neighboring cell.). Once selected, the relative velocity between the particle pair along with the effective diameters (using the Variable Hard Sphere model equation 1.12) of each particle are then calculated.

Finally, the product of the relative velocity and collision cross-section is determined for the selected particle pair and normalized using, $C_{\max}$, the maximum collision product. This relationship is expressed as,

$$\frac{\sigma v_r}{C_{\max}} \quad (3.3)$$

where σ is cross-section and v_r , the relative velocity of the pair. Using a random uniform deviate as a threshold acceptance value for equation 3.3, the particle pair is either selected or discarded. If the particle pair is selected then collision exit velocities are determined. However, if the pair is rejected, and we have not yet checked all the potential collisions (i.e., exceeded the maximum number collisions possible), then two new particles are selected and the process is iterated.

Note that Bird has shown that the use of cellular probabilities (i.e., the two step collision selection process) statistically approaches the actual collision probabilities for the system over time if two simple criteria are satisfied. First, the number of particles in a cell must be sufficient to cover the number of collisions. This is the local equilibrium phenomena previously discussed. This criteria can be met by increasing the number of particles simulated in the system or reducing the size of the cell. Second, $C_{\max}$, the maximum collision product, must always be the maximum. If the collision product of two particles is found to exceed this value, then $C_{\max}$ must be increased (it is usually set to the new maximum value) in order to properly normalize the collision probabilities.

Collision Rules

Once a particle pair has been selected, the particle's collision dynamics must be calculated. That is, the exit velocities (the magnitude and direction) of the collision pair must be determined. The method used to simulate the exit velocities of colliding particles is referred to as the simulation's collision rules. In this work (the viscosity simulation experiments), two general types of collision rules were used, Bird's random exit velocity rules, referred to in this paper as the "normal rules," and biased velocity rules, referred to as "go with the flow" (GWTF) rules.

In both types of viscosity simulations, Rayleigh-Bénard and Poiseuille, data runs were taken using Bird's normal collision rules to create baseline comparison runs. Experimental runs were then taken with the same initial parameters used in the baseline runs except for hybrid mixtures of normal and GWTF collisions (defined below). It was found that only small portions of GWTF could be applied to a simulation. When large percentages of GWTF were used the simulation's stability would deteriorate (e.g., cells with no particles would develop). Typically, no more than a 1% mixture of GWTF was ever used.

Due to conservation of momentum, for purely elastic collisions (Figure 3.2), the center of mass velocity and the overall velocity magnitude of a particle collision pair remains unchanged after a collision. However, the relative velocity of each particle pair can vary.

Figure 3.2. The relative velocity, v_{rel} , and center of mass velocity, v_{cm} , of a particle pair (P1 and P2).

The distinguishing difference between Bird's normal collision rule and GWTF is the relative exit velocities of the colliding particles. Using Bird's normal rules (Figure 3.3), the relative velocity of a particle pair is determined randomly such that after the collision, all directions are equally probable (without violating conservation of momentum).

Figure 3.3. The exit velocities of a particle pair after a normal collision (note that in DSMC, particle positions are not changed during the collision, only the velocities are modified).

In GWTF, the relative exit velocities (i.e., unit vectors) are aligned with the center of mass velocity (Figure 3.4). In the unlikely case that the center of mass velocity is zero, GWTF is not employed. Because of the tendency of the center of mass velocity to be the stream flow, the exit velocities tend to “go with the flow” (hence the term).

Figure 3.4. The exit velocities of a particle pair after a GWTF collision.

Note that when GWTF collisions are used, conservation of momentum and energy is still maintained, only the post-collision exit velocities are biased in the direction of the fluid flow.

Flow Configurations

The parameters discussed thus far make up what is referred to as a simulation kernel. That is, the physical particle parameters, dimension parameters, and collision calculations define a complete dilute helium gas simulation environment. However, the configuration of the system (e.g., Poiseuille or Rayleigh-Bénard) is in fact determined by the application of external forces.

Poiseuille Flow

Poiseuille flow, or planar pipe flow as it is often referred to, consists of fluid flowing between two infinitely long parallel boundaries. The fluid flow in these systems is driven by a potential field parallel to the boundaries. In the Poiseuille simulation runs used in this work, the boundaries were 47λ apart and configured as thermal reservoirs held at 409.5K. Total thermalization was applied to particles upon collision with the boundary.

The other two boundaries, perpendicular to the thermal wall, were periodic (thus approximating infinite boundary lengths) and also 47λ apart.

The potential field driving the fluid (gas) flow was a uniform gravitational field. Since the strength of a gravitational field directly affects the average particle velocity (parallel to the thermal boundaries), the applied gravitational acceleration determined the simulation time step. After a few trial iterations, a gravitational acceleration of $1.0 \times 10^5 \text{ m/s}^2$ was applied with a time step of $9.4 \times 10^{-7} \text{ s}$.

Rayleigh-Bénard Flow

In contrast to the Poiseuille configuration (which is primarily driven by a gravitational field), the Rayleigh-Bénard systems have two opposing forces at work, which when properly balanced, induce convection. First there are two parallel thermal reservoir type boundaries (i.e., walls) which maintain a temperature difference or gradient, and inject thermal kinetic energy into the system. And second, there is a gravitational potential field perpendicular to the thermal walls. In order for the particles in the system to remain uniformly distributed, a balance must be maintained by these two forces. If the particle distribution becomes sparse in certain regions, convection is inhibited and computational instabilities may arise due to the absence of particles.

The two forces, gravity and thermal kinetic energy, were balanced by matching the gravitational potential energy exerted on a single particle with the thermal kinetic energy maintained by the temperature gradient. This relationship maintains approximate uniform density throughout the system and is given by,

$$k\Delta T = mgL \quad (3.4)$$

where T is the temperature gradient, k the Boltzmann constant, m the mass of a particle, g the gravitational acceleration, and L the distance between thermal walls.

Note that whenever the temperature gradient between the two walls was increased during a simulation run (in order to induce convection), the gravitational field was automatically adjusted to compensate for the increased kinetic energy. However, the applied time step was always 9.4×10^{-6} s.

In order for convection to develop, the system must exceed its critical Rayleigh number (Ra). This depends on the boundary conditions at the wall (i.e., total thermalization versus a combination of specular and thermalization). Assuming a large aspect ratio (i.e., the ratio between the width of the system and the distance between the thermal walls), the critical Rayleigh number is either 1708 for total thermalization boundaries, or 658 for combination specular and thermalizing boundaries. Since the aspect ratio in this work was unity and the boundaries were a combination of thermalizing and specular, the critical Rayleigh number is approximately 780.¹

Using the critical Rayleigh number, the Rayleigh number equation (1.22) given in Chapter 1, and the Chapman-Enskog theory for simple real gases (hard spheres), the minimum dimensions for convection to develop in a system, can be estimated. For example, assuming uniform density, the coefficient of volume α , is inversely proportional to temperature (i.e., $\alpha = 1/T$); the kinematic viscosity ν is $5/32 \pi \bar{c} \lambda$; and the thermal diffusivity, κ , is $5/2 \nu$. Inserting these relationships into the Rayleigh number equation 1.22, the equation can be rewritten as,

$$R = \frac{256}{125\pi} \left(\frac{\Delta T}{T} \right)^2 \left(\frac{L}{\lambda} \right)^2 \quad (3.5)$$

¹ A. Garcia, in Microscopic Simulations of Complex Flows, ed. M. Mareschal, (New York: Plenum, 1990), p. 182.

where L is the characteristic length, λ the mean free path, ΔT the applied temperature gradient and T , the average temperature.

Therefore, using equation 3.5, assuming $R_c = 780$, and $\Delta T/T$ is on the order of unity, a Rayleigh-Bénard simulation must have at least a 35 mean free paths distance between thermal walls in order for convection to develop. Furthermore, given a mean free path of, 0.03174 m, the distance between the walls must be at least 1.11 m. Thus, our 1.5 m (47λ) system dimensions (between boundaries) are well within convection bounds.

As mentioned above, the two side walls (the non-thermal walls) were purely specular, while the two thermal reservoir walls were simulated with part thermal and part specular boundaries (slightly lowering the critical Rayleigh number of the system). The simulation system was defined to lie on the xy-plane with the x-axis perpendicular to the thermal walls, the y-axis parallel to the thermal walls and the z-axis normal to the xy-plane. Therefore, a particle colliding with a thermal boundary, would have its velocity thermalized in the x and z directions and specular reflected in the y direction.

CHAPTER 4

SIMULATION RESULTS

There were two distinct simulation configuration (flows) runs, Poiseuille and Rayleigh-Bénard. Data for each configuration was first collected using normal Bird collision rules. This provides a base line from which to compare the hybrid data runs (which are a mixture of GWTF and normal collision rules).

Every four time steps a data sample was taken, after 2000 time steps the average of these samples was saved to a binary file. This sample represents the average state of the system per cell over the last 2000 time steps. The binary file was then processed by the VCO2 program to compute spatial averages (over larger cell regions) and extract fluid statistics. Finally, the VCO2 output was in turn saved to an ASCII file and processed by MATLAB (a statistical analysis and plotting tool). Averaging the simulation samples in 2000 time step increments not only reduces statistical scatter but allows for the temporal development of the system to be mapped and the determination of steady state to be made. Note that the simulation of 2.0×10^5 particles for 2000 time steps typically took about 4 hours of CPU time (depending upon the number of collisions calculated per time step) on an IBM RS/6000 workstation. Therefore, with each simulation taking 20,000 to 42,000 time steps, three or more days of run time per simulation were not uncommon.

Poiseuille Flow

Three Poiseuille simulations were run using DSMC. Each run was identically configured except for a varying mixture of GWTF and normal collisions applied to the particles. The first run used no GWTF (i.e., 100% normal collision rules), the second run 0.25% GWTF (99.75% normal), and the third run 1.0% GWTF (99% normal).

In these Poiseuille data runs, the particle flow of the system was driven by a gravitational potential field. Thus, the system reached equilibrium once the particles stopped accelerating and the stream velocity became constant.

Velocity Data

In Figure 4.1 the average velocity of each of the three Poiseuille simulation runs are shown versus time. The simulations were each run for a total of 42,000 time steps with temporally averaged flow samples saved to file every 2000 time steps (i.e., one data point on the plots). The standard deviation of the last five saved data points (10,000 time steps) was taken and applied as error bars to the curves.

Figure 4.1. Three Poiseuille runs approaching steady state stream velocity. The only parameter variation between the curves is the amount of GWTF used during the collision calculations.

All three curves (velocity vs. time) eventually converge to steady state. This indicates that the system is stable even with the addition of small amounts of GWTF, and that the viscosity must therefore be finite (i.e., not zero). Note that the runs that used larger percentages of GWTF reached higher equilibrium velocities but took more time steps to get there.

In Figure 4.2 three curves, one from each of the Poiseuille data runs, are shown. Each curve represents the average velocity swath (100 swaths, each one cell wide and 100 cells long) across the steady state stream flow (perpendicular to the walls).

Figure 4.2. The steady state, spatially averaged velocity swaths (parallel to the flow) of three Poiseuille simulation runs under various collision rule combinations.

As previously discussed, viscosity is the measure of friction (momentum transfer) between parallel flow streams, the maximum stream velocity of a system at steady state is therefore inversely proportional to its viscosity. Since the viscosity of the system at steady state is expected to produce a parabolic velocity swath (see equation 1.21), the effect that a small percentage of GWTF has on viscosity can be computed by fitting the velocity swath data in Figure 4.2 to a polynomial. The coefficient of the second order term of the polynomial, C_2 , is thus related to the viscosity of the system as,

$$C_2 = \frac{\rho g}{2\eta} \quad (4.1)$$

where, η , is the dynamic viscosity, ρ the density, and g the gravitational acceleration. Using 1.4×10^{20} particles/m³ as the density, 1.0×10^5 m/s² as the gravitational acceleration, the viscosity is calculated for each of the three velocity swath curves in Figure 4.6, and shown in Table 4.1. Note that in Figure 4.2, although the pure normal rule curve (bottom) fits the typical parabolic shape, the runs using GWTF (top and middle) are better represented by a higher order fit.

Table 4.1. The computed viscosity for Poiseuille runs.

Run	Viscosity (η)
Normal collisions	2.712×10^{-5} N s/m ²
0.25% GWTF collisions	2.273×10^{-5} N s/m ²
1.0% GWTF collisions	1.580×10^{-5} N s/m ²

Temperature and Density Data

Besides the stream flow computations, the macroscopic parameters temperature and density were also computed. Although the velocity stream profiles indicate a slight variation in the flow when a mixture of GWTF is included in the simulation, the density and temperature plots show a more dramatic difference.

Figures 4.3 and 4.4 show the temperature and density swaths (spatially averaged) of the normal collision system shown as the bottom curve in Figure 4.2 while Figures 4.5 and 4.6 show the corresponding temperature and density plots for the 1% GWTF mixture (the top curve in Figure 4.2).

Figure 4.3. The steady state, spatially averaged temperature swath (across the stream flow) of a normal rule Poiseuille run (the bottom curve in Figure 4.2).

Figure 4.4. The steady state, spatially averaged density swath (across the stream flow) of a normal rule Poiseuille run (the bottom curve in Figure 4.2).

Figure 4.5. The steady state, spatially averaged temperature swath (across the stream flow) of a 1% GWTF Poiseuille run (the top curve in Figure 4.2).

Figure 4.6. The steady state, spatially averaged density swath (across the stream flow) of a 1% GWTF Poiseuille run (the top curve in Figure 4.2).

As is typically seen in Poiseuille flow, using the normal collision rules a temperature swath across the stream flow produced a rough quartic ($T \propto x^4$) curve (concave up). The total variation between the highest and lowest temperature values was about 50K with the highest temperatures being towards the center of the flow stream. The density using normal collision rules was also quartic ($\rho \propto 1/T$) in shape but concave down and the density variation was only about a factor of one tenth ($9.9 \times 10^{-7} \text{ kg/m}^3$ at the walls and $9.1 \times 10^{-7} \text{ kg/m}^3$ towards the center).

With a 1% GWTF mixture, the temperature and density plots diverged from the norm. Unlike the normal collision run, the density across the swath was not quartic. In fact, the curve was quasi-parabolic in shape and concave down (as opposed to the concave up in the normal case). The variation between the highest and lowest values was

a factor of 2.5 ($1.4 \times 10^{-6} \text{ kg/m}^3$ at the center, $5.7 \times 10^{-7} \text{ kg/m}^3$ at the walls). Note that the center of the stream flow in the GWTF mixture was considerably cooler than the walls (as opposed to hotter center in the normal case) with a variation between the maximum and minimum temperature of about 260K. However, the temperature across the swath (the curve) was consistent with density (i.e., $\rho \propto 1/T$).

Rayleigh-Bénard Flow

Fifteen Rayleigh-Bénard simulations were run using DSMC. Six runs were made using only normal collisions while the other nine runs used a mixture of 1% GWTF and 99% normal collisions.

Besides the collision rules, the only other parameter which varied between the data runs was the temperature gradient between the thermal walls. As shown by the Rayleigh equation (1.22), if all parameters in a system are held constant, then at some critical temperature (a gradient between the walls) the critical Rayleigh number is achieved and a convection phase transition will occur. Therefore, given two identically configured systems (i.e., the same critical Rayleigh number), if convection occurs at the same temperature gradient then the systems have the same viscosity. Conversely, if the two systems do not convect when the applied temperature gradient is the same, then the relative viscosity between the two systems must be different.

Unless the convection transition point is located, it is difficult to determine the specific viscosity value from the Rayleigh equation (equation 1.22). However, once a system passes the phase transition threshold, an increase in the Rayleigh number produced by a higher applied temperature gradient, will increase the speed of the convection flow. Therefore, it is possible to compare the relative viscosity of two systems once the steady state convection speed is calculated.

Velocity Data

Given two systems, one with 1% GWTF (99% normal collisions) and one having exclusively normal collisions, it is possible to compare their relative viscosity. The system using all normal collisions had an applied temperature gradient of 455K while the system using some GWTF had an applied temperature gradient of 136.5K. The convection vector flow of these two systems at equilibrium are shown in Figures 4.7 and 4.10 along with contour plots of the x and y velocity components (Figures 4.8, 4.9, 4.11 and 4.12). Note that each vector represents the spatially averaged particle velocities of 16 cells (4 by 4 cells) using one temporally averaged data sample. Therefore, in order to improve plot readability, the 100 by 100 sample cells used in the simulation are shown as 25 by 25 vectors.

Figure 4.7. The steady state convection flow of a 2-dimensional Rayleigh-Bénard simulation using normal collision rules and a temperature gradient of 455K.

Figure 4.8. The x velocity component (m/s) of Figure 4.7.

Figure 4.9. The y velocity component (m/s) of Figure 4.7.

Figure 4.10. The steady state convection flow of a 2-dimensional Rayleigh-Bénard simulation using 1.0% GWTF rules and a temperature gradient of 136.5K.

Figure 4.11. The x velocity component (m/s) of Figure 4.10.

Figure 4.12. The y velocity component (m/s) of Figure 4.10.

It should be pointed out that although these plots (Figures 4.7 and 4.10) use different collision rules they both have stable and uniform convection flows. That is, the addition of a 1% GWTF collision mixture can produce a stable convection flow.

By observing the temporal development of the two systems over the course of their simulation, steady state can be discerned and its convection speed measured. In Figure 4.13, the convection speed, expressed as the magnitude of the average flow velocity, is shown in 2000 time step increments. The average flow velocity in all of the Rayleigh-Bénard plots at a given sample point was computed by averaging all particle velocities in each simulation cell and then averaging spatially over all cells.

Figure 4.13. The temporal convergence of the two Rayleigh-Bénard simulation runs, shown in Figures 4.7 and 4.10, to steady state.

There are two observations that can be made from this plot (Figure 4.13). First, the steady state convection speed is about the same for both plots (119 m/s for the normal collision run and 116 m/s for the GWTF mixture run) even though the applied temperature gradients between the two runs varies by 318.5K. Therefore, if all other system parameters between the two systems are identical, the GWTF mixture system must have a lower viscosity. Second, the GWTF mixture system exhibits twice the convergence rate to steady state than does the pure normal rule simulation. This is also indicative of a system with less internal friction, that is, a lower viscosity.

By computing the steady state convection speed of all the data runs a two dimensional phase diagram, showing the convection speed versus the applied temperatures, can be constructed. In Figure 4.14 the convection speeds of all the single vortex convection flows (i.e., the first convection phase transition) are plotted versus the

applied temperature gradient. Each data point shown is computed by averaging the last five data samples of a simulation run (i.e., a particular temperature gradient) after it reaches steady state flow. While each sampled data run is computed by first averaging all particle velocities in a cell and then averaging all cells in the simulation space. Note that an average velocity magnitude flow (i.e., the convection speed) of 5 m/s (or less) indicates the absence of a steady state convection flow (speed is not zero due to statistical scatter). While a flow greater than 5 m/s was found to represent the presence of steady state convection flow.

Figure 4.14. A convection phase diagram showing the average convection velocity versus the applied temperature gradient of two 2-dimensional Rayleigh-Bénard simulation systems.

A first order extrapolation of a convection phase diagram can be made for each rule type (normal and 1% GWTF) by connecting the data samples (i.e., data runs) in Figure 4.14. Unfortunately, due to the computational expense of each data run, there is a

scarcity of data points. Therefore, the actual phase transition curve drawn between the data samples may not correctly predict specific convection transition points. At best only a temperature range can be inferred between any two samples.

Figure 4.15. The steady state velocity convection flow of a 2-dimensional Rayleigh-Bénard Simulation using normal rules and a temperature gradient of 546K.

Figure 4.14 only shows the first convection phase transitions. When higher temperatures are applied, a bifurcation occurs and a double convection vortex (the second convection phase transition) develops. During the early simulation runs, the effect that 1% GWTF collisions would have on a system was not known. Since a stable convection flow was found to develop using normal collisions with an applied temperature gradient of 546K (shown in Figure 4.15), the same simulation was run again using a 1% GWTF collision mixture. The observed flow pattern that developed (shown in Figure 4.16) had a double convection vortex. This indicates that using a small percentage of GWTF

collisions, a bifurcation transition occurs at temperatures less than if normal collisions were used exclusively. A lower bifurcation transition temperature is thus indicative of a lower system viscosity.

Figure 4.16. The steady state velocity convection flow of a 2-dimensional Rayleigh-Bénard Simulation using 1.0% GWTF rules and a temperature gradient of 546K.

Temperature and Density Data

Although the convection flows observed using the GWTF mixtures are very similar to the pure normal rule flows, some variations are found in the temperature and density plots. For example, Figure 4.17 and Figure 4.18 show the temperature and density of the convection run shown in Figure 4.7. The contours, in both the temperature and density plots, are indicative of the Rayleigh-Bénard convection. That is, recently heated particles are less dense and tend to rise up one side of the system while cooler particles, with higher densities flow down the other side of the system.

Figure 4.17. The temperature contours (K) of the normal rule Rayleigh-Bénard convection system show in Figure 4.7 (temperature gradient = 455K).

Figure 4.18. The density contours (kg/m^3) of the normal rule Rayleigh-Bénard convection system shown in Figure 4.7 (temperature gradient = 455K).

In comparison, Figures 4.19 and 4.20 show the temperature and density of the GWTF mixture system at steady state convection in Figure 4.10. Note that these plots have different applied temperature gradients, but their convection flow is approximately the same.

Figure 4.19. The temperature contours (K) of the 1% GWTF rule Rayleigh-Bénard convection system shown in Figure 4.10 (temperature gradient = 136.5K).

The temperature and density plots using a mixture of GWTF and normal collision rules are generally similar in appearance (to Figures 4.17 and 4.18), but differ in a few areas. First, there is a clumping of particles as they fall during the cooling portion of the convection cycle. This can be seen in the density plot as a small high density band toward the bottom of the system and as a large temperature drop (high packing of contours) in the same relative spatial location in the temperature plot. Second, there is a high temperature jump observed around the thermal boundaries. In fact, the entire system

tends to cool significantly faster toward the center regions of the system (compared with the normal runs).

Figure 4.20. The density contours (kg/m^3) of the 1% GWTF rule Rayleigh-Bénard convection system show in Figure 4.10 (temperature gradient = 136.5K).

Discussion

The data presented above shows a number of anomalies when comparing normal fluid (simulations using purely normal collision rules) and GWTF fluid (simulations using a mixture of GWTF and normal collision rules).

In the Poiseuille runs, increasing mixtures of GWTF lowered the apparent viscosity of the fluid. Using the maximum GWTF mixture of 1%, the viscosity dropped to about one half that of the normal fluid. However, the GWTF velocity profile was not purely parabolic. Theory and the normal fluid simulation runs presented in this work, both

support a quadratic velocity profile, but the GWTF velocity profile deviated slightly from the expected parabolic form.

More significantly, the temperature plots in the Poiseuille GWTF simulations varied quite significantly from the normal fluid. From the Navier-Stokes equations, the temperature is expected to increase due to the velocity gradient inside the fluid (viscous shear heating). One would therefore expect the temperature to increase towards the center of the flow and follow a quartic (i.e., $T \propto x^4$) profile. However, in the GWTF fluid the temperature actually decreased in the center. This cooling process is due to the transformation of thermal energy into kinetic energy in the fluid flow as the relative velocities of colliding GWTF particles align with the center of mass. Since the cooling process is correlated to the established flow, we referred to this phenomenon as “entrainment cooling.” Note that the GWTF density plots also differed from those of the normal fluid, but since density is inversely proportional to temperature, these results are consistent.

The velocity profiles in the convection flows observed using GWTF collision rules were very similar to the normal fluid simulation runs. However, temperature and density anomalies, primarily due to entrainment cooling, were also seen in the Rayleigh-Bénard GWTF simulations. The most prominent difference was the onset of convection—using GWTF, convection occurred at a much lower temperature. Since the viscosity of a fluid is correlated to the convection process, the onset of convection at lower temperature gradients implies a lower fluid viscosity. Furthermore, the phase diagram (flow speed versus gradient temperature) for GWTF was similar except for a temperature offset (see Figure 4.14). In fact, the onset of convection may occur as low as $\Delta T = 0$; Direct application of the Rayleigh number formula therefore implies that the viscosity of the GWTF system approaches zero. However, this disagrees with Poiseuille viscosity calculations which indicated that the viscosity was lower but only by a factor of two.

Obviously, the Poiseuille and Rayleigh-Bénard viscosity calculations do not tell the whole story. Transport processes in the GWTF fluid appear to be more complex than in the normal dilute gas. A possibility is that both viscosity, η , and thermal diffusivity, κ , are dependent on velocity. Indications of a velocity dependence can be seen in the GWTF collision process. In GWTF, if there is no net velocity (i.e., zero stream flow), the center of mass velocity is isotropically distributed and the biased collisions are averaged out. In this case, GWTF fluids are indistinguishable from normal fluids. This is readily observed in the Rayleigh-Bénard convection flows. That is, when no temperature gradient is applied to a system, convection can not develop (there is no net thermal energy flow), and the GWTF collisions have no net effect. However, when a temperature gradient is applied, convection starts and the GWTF collisions quickly entrain the flow. In fact, convection always develops quickly in the GWTF systems (see Figure 4.13).

Finally, when comparing GWTF fluid with normal fluid, the direction of the convection flow should not be overlooked. In Figure 4.7 (normal fluid), the flow was in the counter clockwise direction, while in Figure 4.10 (GWTF fluid) the flow moves clockwise. Therefore, even though the velocity contour plots look similar, the fact that the flows convect in opposite directions, indicates significant differences in the fluid properties. In normal fluid convection, the flow is maintained by the balance of thermally induced kinetic energy and gravitational acceleration. Due to the gravitational influence on each particle, the downward flow (toward the thermal wall) tends to be faster than the thermally induced upward flow. That is, fluid flow in a normal fluid is established by the gravitational acceleration. However, in GWTF fluid, entrainment dictates the flow. That is, after thermalization, particles gain instantaneous velocity and quickly define an upward flow. As particle collisions occur in this flow, energy is sapped from the thermal kinetic energy and pumped into the flow, thus further reinforcing it. Gravitational acceleration on the other hand, adds energy temporally (in small but

uniform increments) and therefore does not have a major effect on the established flow.

This is consistent with the GWTF convection plots which show a faster flow in the thermally heated portion of the convection flow. Further, since the thermally induced upward flow in GWTF fluids tend to dominate the convection cycle, particle clumping is observed as high density regions (Figure 4.20).

CHAPTER 5

CONCLUSIONS

Recently, considerable computational fluid dynamics work has focused on the study of turbulence. In order for turbulence to develop in a fluid system, the Reynolds number, $Re = uL/\nu$, must be greater than about 10^3 . Obtaining such a high Reynolds number using particle simulation methods is currently out of reach. To understand why, take a typical but realistic three-dimensional fluid system. The viscosity of a real gas, using the hard sphere approximation is, $\nu \approx 1/2 \lambda a$, where a is the sound speed, and λ the mean free path. Taking the stream velocity to be about half the speed of sound, $u \approx a/2$ (in order to avoid shock waves), the minimum size of a fluid system in which turbulence will develop is $L \approx 10^3 \lambda$. And since the system is three-dimensional, the volume is, $V = L^3 \approx 10^9 \lambda^3$. Even if the simulation uses only 10 particles per cubic mean free path, 10^{10} total particles are required. Unfortunately, the maximum number of particles that can be simulated using today's super computers is around 10^7 , or three orders of magnitude less.¹

Even with the current rapid advancement in computer hardware, one would have to wait three decades before reaching 10^{10} particles. Simulating turbulence though is not a lost cause; instead of waiting on hardware, we can seek an algorithmic solution. From the Reynolds equation we note that viscosity is proportional to characteristic length for a fixed Reynolds number, $\nu \propto L$, and the number of particles in a system, N , is proportional to the cube of characteristic length, $L^3 \propto N$. Therefore, for every order of magnitude that viscosity is lowered, three corresponding orders of magnitude are reduced from the total number of particles. By lowering viscosity one order of magnitude, it is then possible to simulate a turbulent flow with only 10^7 particles. Thus the work

¹D. Baganoff, (personal communication), June 1992.

presented focused dogmatically on particle collision rules and their effects on the viscosity of a dilute gas; specifically, finding collision rules which lower it.

Overall though, the attempt to lower viscosity using GWTF collision rules has yielded promising yet mixed results. On the plus side, GWTF collision rules lowered the apparent viscosity in Poiseuille flow by about a factor of two. The Rayleigh-Bénard runs also showed an apparent viscosity change using GWTF collision rules. The reduction in viscosity, in this case, becoming evident through the extreme instability of the system (i.e., convection occurring even for infinitesimal temperature gradients). Unfortunately, the flows did not behave like normal fluids, as was readily evident in the density and temperature measurements in both the Poiseuille and Rayleigh-Bénard flows.

“Entrainment cooling” (the transfer of thermal kinetic energy to the flow stream) significantly changed the physics of the fluid, thus making viscosity comparisons impossible. In brief, the GWTF fluid has a considerably more complicated rheology.

Although the GWTF fluids did not correctly simulate real gases, the original motivating ideas behind GWTF are still valid. Since viscosity is a measure of the momentum transfer perpendicular to a stream flow, GWTF was specifically designed to reduce this transfer. Note that momentum and energy are conserved in GWTF collisions. Yet momentum transfer is suppressed in GWTF by using asymmetry so that transport perpendicular to the stream flow can be minimized. Normal collision rules are symmetrical. That is, on average, the exit velocities (momentum transport) of the particles are equally probable in all directions and interactions between streams in a flow are frequent.

So why did GWTF fail? In part this answer is linked to the asymmetry of the GWTF collisions. The symmetry of the normal collision rules gives time reversibility while the asymmetry of the GWTF collision rules breaks time reversibility. This can be made readily apparent in a simple example. For normal collisions, given two unlabeled lists of

particle velocities—one showing the pre-collision velocities and one showing the post-collision velocities—it is impossible to determine which list is which. In contrast, if the two lists represent GWTF pre-collision and post-collision velocities, the asymmetry of the collision exit velocities is apparent in one of the lists. Therefore, from a microscopic perspective, the interactions must have occurred in a set order (i.e., not time reversible). The fact that the normal collision velocities can not be categorized, makes the lists (on average) indistinguishable and thus time reversible.

Do future collision schemes (designed to reduce viscosity) therefore not only have to maintain the particle level physics—conservation of momentum and energy—but also maintain the time reversibility of the fluid on a microscopic scale? Not necessarily since the macroscopic dynamics are not time reversible (e.g., entropy increases). However, we do have to construct collision schemes with rheologies which do not differ excessively from ordinary fluids. A possible candidate (which is also not time reversible) would be a hybrid GWTF rule. Where normal GWTF applies the collision asymmetry rule to a random percentage of particles, the hybrid rule would only apply the asymmetrical collision rules to particles whose relative velocity was perpendicular to the center of mass. This would reduce the number of particles affected by GWTF to only those which are moving significantly out of the flow (i.e., the particles which produce the most drag).

Another possibility is the “cross rule” suggested by B. Alder. In this collision scheme, the exit velocities of the colliding particles are always selected to be perpendicular to the pre-collision relative velocity. Although work is still in progress, preliminary tests show that the “cross rule” does lower the viscosity of a fluid, but not as much as GWTF.

Finally, it is important to keep this work in proper perspective. Although GWTF did not correctly model all the physics of real fluids, it does provide some interesting fluid dynamics. The insight and experience gained with this work will certainly help the

design of the next generation collision rule, as the pursuit of lower viscosity is continued. Of note, when the lattice gas models were first developed, they also had a number of physics related problems (anisotropy, density dependent advection, no temperature, etc.). Eventually these problems were addressed and many have been overcome. That work led to the development of Lattice Boltzmann models which present a promising future. Bird-type algorithms and Lattice Boltzmann models are converging and we expect the next generation of particle models will successfully simulate fully developed turbulence.

BIBLIOGRAPHY

- Alder, B. J., and T. E. Wainwright. "Phase transition for a hard sphere system." *J. Chem. Phys.* 27 (1957): 1208.
- Allen M. P., and D. J. Tildedley. Computer Simulation of Liquids. Oxford: Clarendon, 1987.
- Anderson, D. A., J. C. Tannehill, and R. H. Pletcher. Computational Fluid Mechanics and Heat Transfer. New York: Hemisphere, 1984.
- Baganoff, D. personal communication, 1992.
- Bird, G. A. Molecular Gas Dynamics. Oxford: Clarendon Press, 1976.
- Bird, G. A. in Microscopic Simulations of Complex Hydrodynamic Phenomena. Proceedings CECAM Conf., Sardinia, 1992.
- Bird, G. A. "Approach to translational equilibrium in a rigid sphere gas." *Phys. Fluids* 6 (1963): 1518.
- Chandrasekhar, S. Hydrodynamic and Hydromagnetic Stability. New York: Dover, 1961.
- Garcia, A. In Microscopic Simulations of Complex Flows. Ed. M. Mareschal. New York: Plenum, 1990.
- Huang, K. Statistical Mechanics. New York: Wiley, 1963.
- Kac, M. Probability Theory and Related Topics in Physical Science. New York: Wiley-Interscience, 1959.
- Landau, L. D., and E. M. Lifshitz. Fluid Mechanics. Oxford: Pergamon Press, 1959.
- Logan, J., and M. Kac. *Phys. Rev. A* 13 (1976): 458.
- Press, William H., Brian P. Flannery, Saul A. Teukolsky, and William T. Vetterling. Numerical Recipes in C. Cambridge: Cambridge University Press, 1988.

APPENDIX

```

/*****
/* DEFINES: The simulation definitions used by VCI2, VCR2 and VCO2 */
/*****
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

/*****
/* Molecular constants: */
/* NUM_MOL is the number of simulated molecules */
/* NUM_SPEC is the number of molecular species */
/* NUM_CELL is the number of collision cell grids in the simulation */
/*****
#define NUM_MOL      200000
#define NUM_SPEC      2
#define NUM_CELL     10000
#define TYPE0         0
#define TYPE1         1

/*****
/* Physical dimension indexing: */
/* UPPER is a constant for indexing the upper surface in some arrays */
/* LOWER is a constant for indexing the lower surface in some arrays */
/*****
#define UPPER        0
#define LOWER        1

#define X_COMP        0
#define Y_COMP        1
#define Z_COMP        2

/*****
/* Indexing constants for SPECIE (species characterization) */
/*****
#define MASS          0
#define DIAM          1
#define VIS_COEF      2
#define TEMP_COEF     3
#define NUM_FRAC      4
#define GAMMA_FUNC    5
#define SPEED          6

/*****
/* Indexing constants for SAMP (cumulative specie variables per cell) */
/*****
#define NUMBER        0
#define X_VEL         1
#define Y_VEL         2
#define SQR_VEL       3
#define UV_PRODUCT    4

```

```

/*****
/* Indexing constants for SURFACE (various specie char on a surface) */
*****/
#define MOL_SUM      0
#define INCI_NORM    1
#define INCI_PAR     2
#define REFL_NORM    3
#define REFL_PAR     4
#define INCI_FLUX    5
#define REFL_FLUX    6

/*****
/* Indexing constants for MOL_LOC & CELL_MOL */
*****/
#define SPEC         1
#define ADDR         0
#define NUM          1

#define TRUE         1
#define FALSE        0

/*****
/* Macro definitions */
*****/
#define      sqr(x)  ((x)*(x))

/*****
/* Portable Random Number Generator Defs */
*****/
#define Modulus      714025
#define IA           1366
#define IC           150889

```

```

/*****
/* EXTERNS: Contains the external varibale definitions for VCI2,      */
/*      VCR2 & VCO2                                                  */
/*****
extern double      CCEL[2];
extern int         CELL[2][NUM_MOL];
extern double      CELL_VOLUME;
extern double      COLLISION[2][NUM_CELL];
extern double      COL_SUM[NUM_SPEC][NUM_SPEC];
extern int         CROSS_REF[NUM_MOL];
extern int         FLOW_SAMP;
extern double      INIT_DEN;
extern int         MOL[2][NUM_CELL];
extern int         MOL_SPEC[NUM_SPEC];
extern int         NEIGHBOR[4][8];
extern double      NUM_SAMP;
extern int         NX;
extern int         NY;
extern double      POS[2][NUM_MOL];
extern int         PRINT_CNT;
extern int         PROG_STOP;
extern double      REAL_MOL;
extern double      REDUCED_MASS[NUM_SPEC][NUM_SPEC];
extern double      REF_TEMP;
extern int         RESTART_FILE;
extern double      SAMP[5][NUM_CELL][NUM_SPEC];
extern double      SPECIE[8][NUM_SPEC];
extern double      SURFACE[7][NUM_SPEC][2];
extern double      SUR_TEMP[2];
extern double      TIME;
extern double      TIME_STEP;
extern double      TIMI;
extern double      VEL[3][NUM_MOL];
extern double      WALL_VEL[2];
extern double      X_CELL_WIDTH;
extern double      X_UP_BOUND;
extern double      X_LOW_BOUND;
extern double      Y_CELL_WIDTH;
extern double      Y_UP_BOUND;
extern double      Y_LOW_BOUND;
extern double      Z_WIDTH;

extern double      TPI, BOLTZ, PI, RANDMAX;
extern double      ranf(), gam();
extern double      WriteRestData(), WriteStatData(), ReadRestData(),
ReadStatData();

```

```

/*****
/* FUNCS: Contains frequently used functions and vars. for VCI2,      */
/*      VCR2 & VCO2                                                  */
/*****
#include "DEFINES.h"

/*****
/* Specific Program Constants                                          */
/*****
double PI          = 3.1415926536;
double TPI         = 6.2831853072;
double RANDMAX     = 2147482647;
double BOLTZ       = 1.380622E-23;

/*****
/* VCR.DAT variable definitions                                       */
/*****
double  ACCEL[2];
int      CELL[2][NUM_MOL];
double  CELL_VOLUME;
double  COLLISION[2][NUM_CELL];
double  COL_SUM[NUM_SPEC][NUM_SPEC];
int      CROSS_REF[NUM_MOL];
int      FLOW_SAMP;
double  INIT_DEN;
int      MOL[2][NUM_CELL];
int      MOL_SPEC[NUM_SPEC];
int      NEIGHBOR[4][8];
double  NUM_SAMP;
int      NX;
int      NY;
double  POS[2][NUM_MOL];
int      PRINT_CNT;
int      PROG_STOP;
double  REAL_MOL;
double  REDUCED_MASS[NUM_SPEC][NUM_SPEC];
double  REF_TEMP;
int      RESTART_FILE;
double  SAMP[5][NUM_CELL][NUM_SPEC];
double  SPECIE[8][NUM_SPEC];
double  SURFACE[7][NUM_SPEC][2];
double  SUR_TEMP[2];
double  TIME;
double  TIME_STEP;
double  TIMI;
double  VEL[3][NUM_MOL];
double  WALL_VEL[2];
double  X_CELL_WIDTH;
double  X_UP_BOUND;
double  X_LOW_BOUND;
double  Y_CELL_WIDTH;
double  Y_UP_BOUND;
double  Y_LOW_BOUND;
double  Z_WIDTH;

```

```

/*****
/* Ranf Function
/*****
double ranf(dum)
long dum;
{
    static long idum, iy, ir[98];
    static int iff=0;
    int j;

    if(dum > 0 || iff == 0)
    {
        iff=1;
        idum = dum;
        if ((idum = (IC - (idum)) % Modulus) < 0) idum = -(idum);
        for (j = 1; j <= 97 ;j++)
        {
            idum = (IA * (idum) + IC) % Modulus;
            ir[j] = (idum);
        }
        idum = (IA * (idum) + IC) % Modulus;
        iy = (idum);
    }
    j = 1 + 97.0 * iy / Modulus;
    iy = ir[j];
    idum = (IA * (idum) + IC) % Modulus;
    ir[j] = (idum);

/* better handle zero, just in case
    if(iy == 0)
    {
        return((double) 1 / Modulus);
    } else
    {
        return((double)iy / Modulus);
    }
}

/*****
/* Gamma Function
/*****
double gam(X)
double X;
{
    double A, Y, GAM;

    A = 1.0;
    Y = X;
    if (Y < 1.0)
    {
        A = A/Y;
    }
    else
    {

```



```

    for(--Y; Y > 1.0; Y--)
    {
        A = A * Y;
    }

    GAM = (1.0 - 0.5748646 * Y + 0.9512363 * Y * Y
           - 0.6998588 * Y * Y * Y);
    GAM = A * (GAM + 0.4245549 * Y * Y * Y * Y
              - 0.101678 * Y * Y * Y * Y * Y);

    return (GAM);
}

/*****
/* Write Stat/Sample Data */
/*****
WriteStatData(filename)
char *filename;
{
    FILE *fp;                                /* file pointer */

    if ((fp = fopen(filename, "wb")) == NULL) {
        printf("Cannot Open File: %s\n", filename);
        exit(1);
    }

    fwrite(&CELL_VOLUME,      sizeof(CELL_VOLUME), 1, fp);
    fwrite(&COL_SUM,          sizeof(COL_SUM),      1, fp);
    fwrite(&NUM_SAMP,          sizeof(NUM_SAMP),      1, fp);
    fwrite(&NX,                sizeof(NX),            1, fp);
    fwrite(&NY,                sizeof(NY),            1, fp);
    fwrite(&REAL_MOL,          sizeof(REAL_MOL),      1, fp);
    fwrite(&REF_TEMP,          sizeof(REF_TEMP),      1, fp);
    fwrite(&SAMP,              sizeof(SAMP),          1, fp);
    fwrite(&SPECIE,            sizeof(SPECIE),        1, fp);
    fwrite(&SURFACE,           sizeof(SURFACE),       1, fp);
    fwrite(&TIME,              sizeof(TIME),          1, fp);
    fwrite(&TIME_STEP,         sizeof(TIME_STEP),     1, fp);
    fwrite(&TIMI,              sizeof(TIMI),          1, fp);

    fclose(fp);
}

/*****
/* Write Start/Restart Data */
/*****
WriteRestData(filename)
char *filename;
{
    FILE *fp;                                /* file pointer */

    if ((fp = fopen(filename, "wb")) == NULL) {

```

```

    printf("Cannot Open File: %s\n",filename);
    exit(1);
}

fwrite(ACCEL,          sizeof(ACCEL),      1,    fp);
fwrite(CELL,          sizeof(CELL),        1,    fp);
fwrite(&CELL_VOLUME,  sizeof(CELL_VOLUME), 1,    fp);
fwrite(COLLISION,     sizeof(COLLISION),    1,    fp);
fwrite(COL_SUM,       sizeof(COL_SUM),      1,    fp);
fwrite(CROSS_REF,     sizeof(CROSS_REF),    1,    fp);
fwrite(&FLOW_SAMP,    sizeof(FLOW_SAMP),    1,    fp);
fwrite(&INIT_DEN,     sizeof(INIT_DEN),     1,    fp);
fwrite(MOL,          sizeof(MOL),          1,    fp);
fwrite(MOL_SPEC,      sizeof(MOL_SPEC),     1,    fp);
fwrite(NEIGHBOR,      sizeof(NEIGHBOR),     1,    fp);
fwrite(&NUM_SAMP,     sizeof(NUM_SAMP),     1,    fp);
fwrite(&NX,          sizeof(NX),           1,    fp);
fwrite(&NY,          sizeof(NY),           1,    fp);
fwrite(POS,          sizeof(POS),          1,    fp);
fwrite(&PRINT_CNT,   sizeof(PRINT_CNT),     1,    fp);
fwrite(&PROG_STOP,   sizeof(PROG_STOP),     1,    fp);
fwrite(&REAL_MOL,    sizeof(REAL_MOL),      1,    fp);
fwrite(REDUCED_MASS,  sizeof(REDUCED_MASS), 1,    fp);
fwrite(&REF_TEMP,    sizeof(REF_TEMP),      1,    fp);
fwrite(&RESTART_FILE, sizeof(RESTART_FILE), 1,    fp);
fwrite(SAMP,         sizeof(SAMP),         1,    fp);
fwrite(SPECIE,       sizeof(SPECIE),       1,    fp);
fwrite(SURFACE,      sizeof(SURFACE),      1,    fp);
fwrite(SUR_TEMP,     sizeof(SUR_TEMP),     1,    fp);
fwrite(&TIME,        sizeof(TIME),         1,    fp);
fwrite(&TIME_STEP,   sizeof(TIME_STEP),     1,    fp);
fwrite(&TIMI,        sizeof(TIMI),         1,    fp);
fwrite(VEL,          sizeof(VEL),          1,    fp);
fwrite(WALL_VEL,     sizeof(WALL_VEL),     1,    fp);
fwrite(&X_CELL_WIDTH, sizeof(X_CELL_WIDTH), 1,    fp);
fwrite(&X_UP_BOUND,  sizeof(X_UP_BOUND),    1,    fp);
fwrite(&X_LOW_BOUND, sizeof(X_LOW_BOUND),    1,    fp);
fwrite(&Y_CELL_WIDTH, sizeof(Y_CELL_WIDTH), 1,    fp);
fwrite(&Y_UP_BOUND,  sizeof(Y_UP_BOUND),    1,    fp);
fwrite(&Y_LOW_BOUND, sizeof(Y_LOW_BOUND),    1,    fp);
fwrite(&Z_WIDTH,     sizeof(Z_WIDTH),       1,    fp);

fclose(fp);
}

/*****
/* Read Stat/Sample Data
*****/
ReadStatData(filename)
char *filename;
{
    FILE *fp;                                /* file pointer */

```

```

if ((fp = fopen(filename, "rb")) == NULL) {
    printf("Cannot Open File: %s\n", filename);
    exit(1);
}

fread(&CELL_VOLUME,    sizeof(CELL_VOLUME),    1,    fp);
fread(COL_SUM,        sizeof(COL_SUM),        1,    fp);
fread(&NUM_SAMP,      sizeof(NUM_SAMP),      1,    fp);
fread(&NX,            sizeof(NX),            1,    fp);
fread(&NY,            sizeof(NY),            1,    fp);
fread(&REAL_MOL,      sizeof(REAL_MOL),      1,    fp);
fread(&REF_TEMP,      sizeof(REF_TEMP),      1,    fp);
fread(SAMP,           sizeof(SAMP),           1,    fp);
fread(SPECIE,         sizeof(SPECIE),         1,    fp);
fread(SURFACE,        sizeof(SURFACE),        1,    fp);
fread(&TIME,          sizeof(TIME),          1,    fp);
fread(&TIME_STEP,     sizeof(TIME_STEP),     1,    fp);
fread(&TIMI,          sizeof(TIMI),          1,    fp);

fclose(fp);
}

/*****
/* Read Start/Restart Data                                     */
*****/
ReadRestartData(filename)
char *filename;
{
    FILE *fp;                                           /* file pointer */

    if ((fp = fopen(filename, "rb")) == NULL) {
        printf("Cannot Open File: %s\n", filename);
        exit(1);
    }

    fread(ACCEL,        sizeof(ACCEL),        1,    fp);
    fread(CELL,          sizeof(CELL),          1,    fp);
    fread(&CELL_VOLUME, sizeof(CELL_VOLUME),    1,    fp);
    fread(COLLISION,     sizeof(COLLISION),     1,    fp);
    fread(COL_SUM,       sizeof(COL_SUM),       1,    fp);
    fread(CROSS_REF,     sizeof(CROSS_REF),     1,    fp);
    fread(&FLOW_SAMP,   sizeof(FLOW_SAMP),     1,    fp);
    fread(&INIT_DEN,    sizeof(INIT_DEN),      1,    fp);
    fread(MOL,           sizeof(MOL),           1,    fp);
    fread(MOL_SPEC,      sizeof(MOL_SPEC),      1,    fp);
    fread(NEIGHBOR,      sizeof(NEIGHBOR),      1,    fp);
    fread(&NUM_SAMP,    sizeof(NUM_SAMP),      1,    fp);
    fread(&NX,          sizeof(NX),            1,    fp);
    fread(&NY,          sizeof(NY),            1,    fp);
    fread(POS,           sizeof(POS),           1,    fp);
    fread(&PRINT_CNT,   sizeof(PRINT_CNT),     1,    fp);
    fread(&PROG_STOP,   sizeof(PROG_STOP),     1,    fp);
    fread(&REAL_MOL,    sizeof(REAL_MOL),     1,    fp);

```

```

fread(REDUCED_MASS,      sizeof(REDUCED_MASS), 1, fp);
fread(&REF_TEMP,        sizeof(REF_TEMP), 1, fp);
fread(&RESTART_FILE,    sizeof(RESTART_FILE), 1, fp);
fread(SAMP,             sizeof(SAMP), 1, fp);
fread(SPECIE,          sizeof(SPECIE), 1, fp);
fread(SURFACE,         sizeof(SURFACE), 1, fp);
fread(SUR_TEMP,        sizeof(SUR_TEMP), 1, fp);
fread(&TIME,           sizeof(TIME), 1, fp);
fread(&TIME_STEP,      sizeof(TIME_STEP), 1, fp);
fread(&TIMI,          sizeof(TIMI), 1, fp);
fread(VEL,             sizeof(VEL), 1, fp);
fread(WALL_VEL,        sizeof(WALL_VEL), 1, fp);
fread(&X_CELL_WIDTH,   sizeof(X_CELL_WIDTH), 1, fp);
fread(&X_UP_BOUND,     sizeof(X_UP_BOUND), 1, fp);
fread(&X_LOW_BOUND,    sizeof(X_LOW_BOUND), 1, fp);
fread(&Y_CELL_WIDTH,   sizeof(Y_CELL_WIDTH), 1, fp);
fread(&Y_UP_BOUND,     sizeof(Y_UP_BOUND), 1, fp);
fread(&Y_LOW_BOUND,    sizeof(Y_LOW_BOUND), 1, fp);
fread(&Z_WIDTH,        sizeof(Z_WIDTH), 1, fp);

fclose(fp);
}

```

```

/*****
/* VCI2 PROGRAM: Data, setting and initialization for 2-d simulation */
*****/

#include "DEFINES.h"
#include "EXTERN.h"

/*****
/* Main */
*****/
main(argc, argv)
int   argc;
char *argv[];
{
    int      n, m, l, k, j;
    double   Gravity, wall_vel, TempAdjust;

/***** SET DATA (All quantities are in base SI units *****/

/* INIT_DEN is the number density at the initial gas temp SUR_TEMP */
INIT_DEN      = 1.4E20;

/* SUR_TEMP is the temperature of the surfaces and the initial gas */
if (argc < 3)
{
    TempAdjust = 273.0 / 2.0;
} else
{
    TempAdjust = strtod(argv[2], (char **) NULL) / 2.0;
}

SUR_TEMP[UPPER] = 409.5 - TempAdjust;
SUR_TEMP[LOWER] = 409.5 + TempAdjust;
printf("\nTEMP GRADIENT = %g to %g", SUR_TEMP[LOWER], SUR_TEMP[UPPER]);

/* WALL_VEL is the velocity of the walls */
WALL_VEL[UPPER] = 0.0;
WALL_VEL[LOWER] = 0.0;

/* Species 0 is helium (TYPE0) */
SPECIE[MASS][TYPE0] = 6.645E-27;
SPECIE[VIS_COEF][TYPE0] = 1.9604E-5;
SPECIE[TEMP_COEF][TYPE0] = 0.657;
SPECIE[NUM_FRAC][TYPE0] = 1.0;

/* Species 1 is argon (TYPE1) */
SPECIE[MASS][TYPE1] = 6.63E-26;
SPECIE[VIS_COEF][TYPE1] = 2.2275E-5;
SPECIE[TEMP_COEF][TYPE1] = 0.811;
SPECIE[NUM_FRAC][TYPE1] = 0.0;

/* The simulation cell dimensions (in meters) */
X_CELL_WIDTH = 0.015;
Y_CELL_WIDTH = 0.015;
X_UP_BOUND = 1.50;

```

```

Y_UP_BOUND      = 1.50;
X_LOW_BOUND     = 0.0;
Y_LOW_BOUND     = 0.0;
Z_WIDTH        = 1.0;

/* acceleration declarations */
/* Gravity parameter adjusts for different temp. gradients */
/* R-B flow */
Gravity          = (BOLTZ * (SUR_TEMP[UPPER] - SUR_TEMP[LOWER]))
                  / ((Y_UP_BOUND - Y_LOW_BOUND) * SPECIE[MASS][TYPE0]);
ACCEL[X_COMP]    = 0.0;
ACCEL[Y_COMP]    = Gravity;
printf("\nGRAVITY = %g", Gravity);

/* the reference temperature for the viscosity coefficients */
REF_TEMP        = 293.0;

/* the time step for the uncoupling of molecular motion and collisions*/
TIME_STEP       = 9.4E-6;

/* the number of time steps between flow samples */
FLOW_SAMP       = 4;

/* the number of flow samples between updates of the restart file */
RESTART_FILE    = 25;

/* the number of file updates until the program STOPS (this is
/* redundant when the programs are run interactively)
PROG_STOP = 20;

/* Calculate number of cells along each axis
NX = (int)((X_UP_BOUND-X_LOW_BOUND)/X_CELL_WIDTH);
NY = (int)((Y_UP_BOUND-Y_LOW_BOUND)/Y_CELL_WIDTH);
printf("\nNUM_CELLS = %d\n", NX * NY);

if ((int)(NX * NY) != NUM_CELL)
{
    printf("\n***WARNING***\nThe physical dimensions do not correspond
        to the cell count.\n");
}

/* Determine neighbor x and y cell adjustments
/* 7 6 5
/* 0 X 4
/* 1 2 3

NEIGHBOR[Y_COMP][0] = NEIGHBOR[X_COMP][2] = 0;
NEIGHBOR[Y_COMP][4] = NEIGHBOR[X_COMP][6] = 0;

NEIGHBOR[X_COMP][0] = NEIGHBOR[X_COMP][1] = NEIGHBOR[Y_COMP][1] = -1;
NEIGHBOR[Y_COMP][2] = NEIGHBOR[Y_COMP][3] = NEIGHBOR[X_COMP][7] = -1;

NEIGHBOR[X_COMP][3] = NEIGHBOR[X_COMP][4] = NEIGHBOR[X_COMP][5] = 1;
NEIGHBOR[Y_COMP][5] = NEIGHBOR[Y_COMP][6] = NEIGHBOR[Y_COMP][7] = 1;

```

```

/* The physical time TIME is initialized to zero */
TIME = 0.0

/***** END of DATA *****/

/* REAL_MOL is the number of real molecules represented by each */
/* of the simulated molecules */
REAL_MOL = INIT_DEN * Z_WIDTH * (Y_UP_BOUND-Y_LOW_BOUND)
          * (X_UP_BOUND-X_LOW_BOUND) / NUM_MOL;

/* CELL_VOLUME is the cell volume */
CELL_VOLUME = Z_WIDTH * (Y_UP_BOUND-Y_LOW_BOUND)
              * (X_UP_BOUND-X_LOW_BOUND) / NUM_CELL;

/* PRINT_CNT is a counter for the resetting of the restart files */
PRINT_CNT= 0;

MOL_SPEC[NUM_SPEC-1] = NUM_MOL;

for (n = 0; n < NUM_SPEC; n++)
{
    if (n < NUM_SPEC-1)
    {
        MOL_SPEC[n] = (int)(NUM_MOL * SPECIE[NUM_FRAC][n]);
        MOL_SPEC[NUM_SPEC-1] = MOL_SPEC[NUM_SPEC-1] - MOL_SPEC[n];
    }

    for (m = 0; m < NUM_SPEC; m++)
    {
        REDUCED_MASS[n][m] = (SPECIE[MASS][m] / (SPECIE[MASS][m]
          + SPECIE[MASS][n])) * SPECIE[MASS][n];
        COL_SUM[n][m] = 0.0;
    }
}

/* Molecular properties */
SPECIE[DIAM][n] = sqrt(15.0 * sqrt(BOLTZ)
    * sqrt(SPECIE[MASS][n] * REF_TEMP/PI)
    / (2.0 * (5.0-2.0 *SPECIE[TEMP_COEF][n])
    * (7.0 - 2.0 * SPECIE[TEMP_COEF][n])
    * SPECIE[VIS_COEF][n]));
SPECIE[GAMMA_FUNC][n] = gam(2.5 - SPECIE[TEMP_COEF][n]);
SPECIE[SPEED+UPPER][n] = sqrt(2.0 * BOLTZ * SUR_TEMP[UPPER]
    / SPECIE[MASS][n]);
SPECIE[SPEED+LOWER][n] = sqrt(2.0 * BOLTZ * SUR_TEMP[LOWER]
    / SPECIE[MASS][n]);
}

if (fabs(WALL_VEL[UPPER]) > fabs(WALL_VEL[LOWER]))
{
    wall_vel = WALL_VEL[UPPER];
}
else
{
    wall_vel = WALL_VEL[LOWER];
}

```

```

/* This is a reasonable initial value for the collisional product */
for (n = 0; n < NUM_CELL; n++)
{
    COLLISION[0][n] = 0.0;
    COLLISION[1][n] = PI * sqr(SPECIE[DIAM][TYPE0])
        * (3.0 * ((SPECIE[SPEED+UPPER][TYPE0]
            + SPECIE[SPEED+LOWER][TYPE0])/2.0) + wall_vel);
}

/*****
/* Set the initial gas characteristics */
/*****
for (l = 0, n = 0; l < NUM_SPEC; l++)
{
    k = MOL_SPEC[l];
    for (m = 0; m < k; m++,n++)
    {
        POS[Y_COMP][n] = Y_LOW_BOUND + ranf(0)
            * (Y_UP_BOUND - Y_LOW_BOUND);
        POS[X_COMP][n] = X_LOW_BOUND + ranf(0)
            * (X_UP_BOUND - X_LOW_BOUND);

        for (j = 0; j < 3; j++)
        {
            VEL[j][n] = sqrt(fabs(log(ranf(0)))) * sin(TPI * ranf(0))
                * ((SPECIE[SPEED+UPPER][l]
                    + SPECIE[SPEED+LOWER][l]) / 2.0);
        }

        CELL[ADDR][n] = (int)((int)((POS[Y_COMP][n]
            - Y_LOW_BOUND) / Y_CELL_WIDTH)) * NX
            + ((POS[X_COMP][n] - X_LOW_BOUND)
                / X_CELL_WIDTH));
        CELL[SPEC][n] = 1;
    }
}

/* Generate the start file */
if(argc > 1)
{
    WriteRestData(argv[1]);
} else {
    WriteRestData("vcr_rest.dat");
}
}

```



```

/*****
/* VCR2 PROGRAM: This program computes the flow
*****/

#include "DEFINES.h"
#include "EXTERNS.h"

/*****
/* MAIN
*****/
main(argc, argv)
int argc;
char *argv[];
{
    double a_temp, b_temp, sqr_rel_vel, mag_rel_vel, sign, mol_total;
    double x_init, y_init, m_diam, l_diam, CVR, ASEL;
    double delta_time, l_mass, m_mass, bound, num_density;
    double CVM, CentMVMag, sqr_time_step, rel_vel[3], cent_mass_vel[3];

    int x_cell, y_cell, x, y, l, m, n, i, j, k, ls, ms;;
    int wall, cell, flow_cnt, restart_cnt, ISEL, KSC, ICS, NSEL;

    if (argc > 3)
    {
        ReadRestData(argv[1]);
    } else {
        printf("\nForm: VCR2 'data_start_file'
                'data_save_file' new_sample(i.e., new)'\n\n");
        exit(1);
    }

    if (!strcmp(argv[3], "new", 1))
    {
        printf("\nNew Sample...\n\n");

        PRINT_CNT = 0;
        NUM_SAMP = 0.0;
        TIMI = TIME;

        for (l = 0; l < NUM_SPEC; l++)
        {
            for (m = 0; m < 7; m++)
            {
                SURFACE[m][1][UPPER] = 0.0;
                SURFACE[m][1][LOWER] = 0.0;

                if (m < 5)
                {
                    for (n = 0; n < NUM_CELL; n++)
                    {
                        SAMP[m][n][1] = 0.0;
                    }
                }
            }
        }
    }
}

```



```

        delta_time = TIME_STEP * fabs((bound - POS[X_COMP][n])
                                     / (x_init - POS[X_COMP][n]));
        VEL[X_COMP][n] = - VEL[X_COMP][n];
        POS[X_COMP][n] = bound + VEL[X_COMP][n] * delta_time;
    }

/* Species 1 molecule strikes a wall, sample incident properties */
/* (in frame of reference moving with the surface velocity WALL_VEL) */

    if ((POS[Y_COMP][n] < Y_LOW_BOUND)
        || (POS[Y_COMP][n] >= Y_UP_BOUND))
    {
        if (POS[Y_COMP][n] < Y_LOW_BOUND)
        {
            wall = LOWER;
            sign = 1.0;
            bound = Y_LOW_BOUND;
        } else
        {
            wall = UPPER;
            sign = -1.0;
            bound = Y_UP_BOUND;
        }

        l = CELL[SPEC][n];

        SURFACE[MOL_SUM][l][wall] = SURFACE[MOL_SUM][l][wall] + 1.0;
        SURFACE[INCI_NORM][l][wall] = SURFACE[INCI_NORM][l][wall]
            - sign * SPECIE[MASS][l] * VEL[Y_COMP][n];
        SURFACE[INCI_PAR][l][wall] = SURFACE[INCI_PAR][l][wall]
            + SPECIE[MASS][l] * (VEL[X_COMP][n] - WALL_VEL[wall]);
        SURFACE[INCI_FLUX][l][wall] = SURFACE[INCI_FLUX][l][wall] + 0.5
            * SPECIE[MASS][l] * (sqr(VEL[X_COMP][n])
            - WALL_VEL[wall]) + sqr(VEL[Y_COMP][n])
            + sqr(VEL[Z_COMP][n]));

        /* delta_time is the time interval for the reflected molecule to move */
        delta_time = TIME_STEP * fabs((bound - POS[Y_COMP][n])
                                     / (y_init - POS[Y_COMP][n]));

        /* Now calculate the reflected velocity components */
        /* comment out X_COMP velocity for slip boundary condition */
        a_temp = TPI * ranf(0);
        b_temp = sqrt(fabs(log(ranf(0)))) * SPECIE[SPEED+wall][l];

        /*
            VEL[X_COMP][n] = b_temp * sin(a_temp) + WALL_VEL[wall];
            VEL[Y_COMP][n] = sign * sqrt(fabs(log(ranf(0))))
                * SPECIE[SPEED+wall][l];
            VEL[Z_COMP][n] = b_temp * cos(a_temp);
        */

        /* Set new position and sample the reflected properties */
        POS[Y_COMP][n] = bound + VEL[Y_COMP][n] * delta_time;
        SURFACE[REFL_NORM][l][wall] = SURFACE[REFL_NORM][l][wall]

```

```

        + sign * SPECIE[MASS][1] * VEL[Y_COMP][n];
    SURFACE[REFL_PAR][1][wall] = SURFACE[REFL_PAR][1][wall]
        - SPECIE[MASS][1]
        * (VEL[X_COMP][n] - WALL_VEL[wall]);
    SURFACE[REFL_FLUX][1][wall] = SURFACE[REFL_FLUX][1][wall] - 0.5
        * SPECIE[MASS][1] * (sqr(VEL[X_COMP][n]
        - WALL_VEL[wall]) + sqr(VEL[Y_COMP][n])
        + sqr(VEL[Z_COMP][n]));
    }

/* Calculate new cell locations if molecule moved over a cell boundary*/
    CELL[ADDR][n] = (int)((int)((POS[Y_COMP][n] - Y_LOW_BOUND)
        / Y_CELL_WIDTH)) * NX
        + ((POS[X_COMP][n] - X_LOW_BOUND)
        / X_CELL_WIDTH));
    }

/*****
/* Set the molecule cross-reference array CROSS_REF */
*****/

/* Reset number of molecules in each cell and calculate start address */
    for (n = 0; n < NUM_CELL; n++)
    {
        MOL[NUM][n] = 0;
    }

    for (n = 0; n < NUM_MOL; n++)
    {
        m = CELL[ADDR][n];
        MOL[NUM][m] = MOL[NUM][m] + 1;
    }

    for (n = 0, m = -1; n < NUM_CELL; n++)
    {
        MOL[ADDR][n] = m;
        m = m + MOL[NUM][n];
        MOL[NUM][n] = 0;
    }

    for (n = 0; n < NUM_MOL; n++)
    {
        m = CELL[ADDR][n];
        MOL[NUM][m] = MOL[NUM][m] + 1;
        j = MOL[ADDR][m] + MOL[NUM][m];
        CROSS_REF[j] = n;
    }

/*****
/* Calculate collisions in loop over cells */
*****/
    for (n = 0; n < NUM_CELL; n++)
    {
        if (NUM_SAMP > 1.0)
        {

```

```

        for (l = 0, num_density = 0.0; l < NUM_SPEC; l++)
        {
            num_density = num_density + SAMP[NUMBER][n][l];
        }
/* This time averages the number density by accounting for          */
/* the number of samples                                           */
num_density = num_density * REAL_MOL / (CELL_VOLUME*NUM_SAMP);
    } else
    {
        num_density = MOL[NUM][n] * REAL_MOL / CELL_VOLUME;
    }

/* ASEL is the number of pair selections                            */
ASEL = 0.5 * MOL[NUM][n] * num_density * COLLISION[1][n]
        * TIME_STEP+COLLISION[0][n];
NSEL = (int)ASEL;
COLLISION[0][n] = ASEL - NSEL;
CVM = COLLISION[1][n];
for (ISEL = 0; ISEL < NSEL; ISEL++)
{
    cell = n;
/* first molecule l is randomly selected molecule from cell n      */
k = (int)(ranf(0) * (MOL[NUM][n]-0.000001)) + MOL[ADDR][n] + 1;
l = m = CROSS_REF[k];
while(l == m){
    if (MOL[NUM][n] < 2) /*there are to few molecules in cell n*/
    {
/* determine cell n location using double indexes (x & y)          */
        y = (int)(n / NX);
        x = n - (int)(y * NX);

/* create cumumaltive sum of neighbor mol. and check for boundaries */
        for(i = 0, mol_total = 0.0; i < 8; i++)
        {
            x_cell = x + NEIGHBOR[X_COMP][i];
            y_cell = y + NEIGHBOR[Y_COMP][i];

/* make neighbors periodic about x axis                             */
            if (x_cell > (NX-1)) x_cell = 0;
            if (x_cell < 0) x_cell = NX-1;

/* check and adjust for neighbors who just happen to be walls      */
            if ((y_cell >= 0) && (y_cell < (NY-1)))
            {
                m = (y_cell * NX) + x_cell;
                NEIGHBOR[3][i] = m;
                mol_total = MOL[NUM][m] + mol_total;
            }
            NEIGHBOR[2][i] = (int)mol_total;
        }
/* find random neighbor molecule                                    */
k = (int)(ranf(0) * mol_total);
/* find molecule's cell                                           */
for(i = 0; k >= NEIGHBOR[2][i];i++) {}
cell = NEIGHBOR[3][i];

```

```

    }
/* pick new molecule in cell */
    k = (int)(ranf(0) * (MOL[NUM][cell] - 0.000001))
        + MOL[ADDR][cell] + 1;

/* second molecule m has been chosen from same cell, if possible */
    m = CROSS_REF[k];
}

/* get species type for selected molecules */
    ls = CELL[SPEC][l];
    ms = CELL[SPEC][m];

/* calculate relative velocities between selected pair */
    for (k = 0; k < 3; k++)
    {
        rel_vel[k] = VEL[k][l] - VEL[k][m];
    }

/* calculate the square and magnitude of the relative velocity */
    sqr_rel_vel = sqr(rel_vel[X_COMP]) + sqr(rel_vel[Y_COMP])
        + sqr(rel_vel[Z_COMP]);
    mag_rel_vel = sqrt(sqr_rel_vel);

/* molecular diameters, cross-section and relative speed */
    a_temp = 2.0 * BOLTZ * REF_TEMP
        / (REDUCED_MASS[ls][ms] * sqr_rel_vel);
    l_diam = SPECIE[DIAM][ls]
        * sqrt(pow(a_temp, SPECIE[TEMP_COEF][ls] - 0.5)
        / SPECIE[GAMMA_FUNC][ls]);
    m_diam = SPECIE[DIAM][ms]
        * sqrt(pow(a_temp, SPECIE[TEMP_COEF][ms] - 0.5)
        / SPECIE[GAMMA_FUNC][ms]);

/* CVR is the product of collision cross-section and relative speed */
    CVR = mag_rel_vel * PI * sqr(l_diam + m_diam) / 4.0;

/* accept pair with probability proportional to CVR */
    if (CVR > CVM) CVM = CVR;
    if (CVR/COLLISION[1][n] >= ranf(0))
    {
        COL_SUM[ls][ms] = COL_SUM[ls][ms] + 1.0;
        COL_SUM[ms][ls] = COL_SUM[ms][ls] + 1.0;
        l_mass = REDUCED_MASS[ls][ms]/SPECIE[MASS][ms];
        m_mass = REDUCED_MASS[ms][ls]/SPECIE[MASS][ls];

/* Compute center of mass velocity and magnitude */
        for (k = 0; k < 3; k++)
        {
            cent_mass_vel[k] = (l_mass*VEL[k][l]) + (m_mass*VEL[k][m]);
        }

        CentMVMag = sqrt(sqr(cent_mass_vel[1])
            + sqr(cent_mass_vel[2]) + sqr(cent_mass_vel[3]));
    }

```

```

/* Normal Rule Check */
    if((ranf(0) < 0.99) || (CentMVMag < 1e-10))
    {
        b_temp = 1.0 - 2.0 * ranf(0);
        a_temp = sqrt(fabs(1.0 - sqr(b_temp)));

/* rel_vel is the components of post-collision relative vel */
        rel_vel[X_COMP] = b_temp * mag_rel_vel;
        b_temp = 2.0 * PI * ranf(0);
        rel_vel[Y_COMP] = a_temp * cos(b_temp) * mag_rel_vel;
        rel_vel[Z_COMP] = a_temp * sin(b_temp) * mag_rel_vel;
    } else /* GWTF */
    {
        for (k=0; k < 3; k++)
        {
            rel_vel[k] = mag_rel_vel * cent_mass_vel[k]/CentMVMag;
        }
    }
/* post-collision velocities */
    for (k = 0; k < 3; k++)
    {
        VEL[k][l] = cent_mass_vel[k] + rel_vel[k] * m_mass;
        VEL[k][m] = cent_mass_vel[k] - rel_vel[k] * l_mass;
    }
}

COLLISION[1][n] = CVM;
}
NUM_SAMP = NUM_SAMP + 1.0;

/*****
/* Sample the flow quantities */
/*****
    for (n = 0; n < NUM_MOL; n++)
    {
/* molecule n is species 1 in cell m */
        m = CELL[ADDR][n];
        l = CELL[SPEC][n];
        SAMP[NUMBER][m][l] = SAMP[NUMBER][m][l] + 1.0;
        SAMP[X_VEL][m][l] = SAMP[X_VEL][m][l] + VEL[X_COMP][n];
        SAMP[Y_VEL][m][l] = SAMP[Y_VEL][m][l] + VEL[Y_COMP][n];
        SAMP[SQR_VEL][m][l] = SAMP[SQR_VEL][m][l] + sqr(VEL[X_COMP][n])
            + sqr(VEL[Y_COMP][n]) + sqr(VEL[Z_COMP][n]);
        SAMP[UV_PRODUCT][m][l] = SAMP[UV_PRODUCT][m][l]
            + (VEL[X_COMP][n] * VEL[Y_COMP][n]);
    }
}
printf("\n%g\t%g\t%g\t%g\t%g",TIME,COL_SUM[TYPE0][TYPE0],
    COL_SUM[TYPE0][TYPE1],COL_SUM[TYPE1][TYPE0],COL_SUM[TYPE1][TYPE1]);

/* save restart and stat data */
    WriteStatData(argv[2]);

```

```
WriteRestData(argv[1]);  
}  
}
```



```

/*****
/* VCO2 PROGRAM: This program writes to ASCII file VCO.dat and computes
/* flow statistics
*****/

#include "DEFINES.h"
#include "EXTERNS.h"

double  heat_inc[2][20], heat_ref[2][20], dum[NUM_SPEC];
double  density[NUM_CELL][20], num_density[NUM_CELL][20];
double  a, SMV, VGSUM[20], temp_sum[20], SM, SMUV, SMU;
double  VVEL[NUM_CELL][20], UVEL[NUM_CELL][20];
double  total_samp, SSSUM[20], TT[NUM_CELL][20], SHST, UVELP, SMCC;
double  pres_inc[2][20], pres_ref[2][20], shear_inc[2][20];
double  shear_ref[2][20];

main(argc, argv)
int  argc;
char *argv[];
{
    int      XAve, YAve, nsum[20], xx, yy, x, y, l, n, wall, k, loop;
    double   TTAve, DenAve, UVelAve, VVelAve;
    FILE      *fpraw_den, *fpraw_u, *fpraw_v, *fpraw_t, *fptext;

    XAve = YAve = 4;

    if ((fptext = fopen("vco2.stats.out", "w")) == NULL)
    {
        printf("Cannot Create %s\n", "vco2.stats.out");
        exit(1);
    }
    if ((fpraw_u = fopen("u.vel.out", "w")) == NULL)
    {
        printf("Cannot Create %s\n", "u.vel.out");
        exit(1);
    }
    if ((fpraw_v = fopen("v.vel.out", "w")) == NULL)
    {
        printf("Cannot Create %s\n", "v.vel.out");
        exit(1);
    }
    if ((fpraw_t = fopen("t.out", "w")) == NULL)
    {
        printf("Cannot Create %s\n", "t.out");
        exit(1);
    }
    if ((fpraw_den = fopen("den.out", "w")) == NULL)
    {
        printf("Cannot Create %s\n", "den.out");
        exit(1);
    }

    if (argc < 2)
    {
        printf("\nWrong number of inputs..., try again.\n\n");
    }
}

```



```

                                shear_ref[wall][loop],heat_ref[wall][loop]));
fprintf(fptext,"\nTotal      \t%g\t%g\t%g\n",
        (pres_inc[wall][loop] + pres_ref[wall][loop]),
        (shear_inc[wall][loop] + shear_ref[wall][loop]),
        (heat_inc[wall][loop] + heat_ref[wall][loop]));
    }

/* The shear stress, velocity gradient, heat flux, and temperature */
/* gradient will be sampled over the central half of the flowfield */
/* (inside the Knudsen layers at the walls) (couette flow only) */

/* Commented out to produce smaller ASCII files */
/*      fprintf(fptext,"\nCell\tX\tY\tDensity\tU-Vel\tV- */
/*      VEL\tTemperature\tShear Stress"); */

    nsum[loop]      = 0;
    SSSUM[loop]     = 0.0;
    VGSUM[loop]     = 0.0;
    temp_sum[loop]  = 0.0;

    for (x = 0; x < NX; x++)
    {
        for (y = 0; y < NY; y++)
        {
            n = (y * NX) + x;

            total_samp = 0.0;
            SM         = 0.0;
            SMU        = 0.0;
            SMV        = 0.0;
            SMCC       = 0.0;
            SMUV       = 0.0;

            for (l = 0; l < NUM_SPEC; l++)
            {
                total_samp = total_samp + SAMP[NUMBER][n][l];
                SM = SM + SPECIE[MASS][l] * SAMP[NUMBER][n][l];
                SMU = SMU + SPECIE[MASS][l] * SAMP[X_VEL][n][l];
                SMV = SMV + SPECIE[MASS][l] * SAMP[Y_VEL][n][l];
                SMCC = SMCC + SPECIE[MASS][l] * SAMP[SQR_VEL][n][l];
                SMUV = SMUV + SPECIE[MASS][l]
                    * SAMP[UV_PRODUCT][n][l];
            }

            num_density[n][loop] = (total_samp * REAL_MOL)
                / (CELL_VOLUME*NUM_SAMP);
            density[n][loop] = num_density[n][loop]
                * (SM / total_samp);
            UVEL[n][loop] = (SMU/total_samp) / (SM/total_samp);
            VVEL[n][loop] = (SMV/total_samp) / (SM/total_samp);
            TT[n][loop] = ((SMCC/total_samp) - (SM/total_samp)
                * sqr(UVEL[n][loop])) / (3.0 * BOLTZ);
            SHST = -num_density[n][loop] * (SMUV/total_samp);

```

```

        if ((y > (int)(NY/4.0)) && (y < (int)(3.0*NY/4.0)))
        {
            nsum[loop]++;
            SSSUM[loop] = SSSUM[loop] + SHST;
            VGSUM[loop] = VGSUM[loop] + (UVEL[n][loop] - UVELP)
                / Y_CELL_WIDTH;
            temp_sum[loop] = temp_sum[loop] + TT[n][loop];
        }
        UVELP = UVEL[n][loop];

/* Commented out to produce smaller ASCII files */
/*      fprintf(fptext, "\n%d\t%d\t%d\t%g\t%f\t%f\t%g\t%g", */
/*              n, x, y, density[n][loop], UVEL[n][loop], */
/*              VVEL[n][loop], TT[n][loop], SHST); */

    }
}

fprintf(fptext, "\nAverage Shear Stress = %g",
        SSSUM[loop]/nsum[loop]);
fprintf(fptext, "\nAverage Velocity Gradient = %f",
        VGSUM[loop]/nsum[loop]);
fprintf(fptext, "\nAverage Temperature = %f",
        temp_sum[loop]/nsum[loop]);
fprintf(fptext, "\nViscosity Coefficient = %g",
        SSSUM[loop]/VGSUM[loop]);
}
fclose(fptext);

/* Output ASCII files for MatLab Analysis, Spatially Averaged */
/* when indicated */
for (y = NY-1; y > 0; y=y-YAve)
{
    for (x = 0; x < NX; x=x+XAve)
    {
        VVelAve = UVelAve = TTAve = DenAve = 0.0;
        for (yy = 0; yy < YAve; yy++)
        {
            for (xx = 0; xx < XAve; xx++)
            {
                /* adjust cell for spatial XxY blocks */
                n = ((y - yy) * NX) + (x + xx);

/* Loop and average over all inputted data runs */
                for (loop = 0; loop < argc-1; loop++)
                {
                    VVelAve = VVelAve + VVEL[n][loop];
                    UVelAve = UVelAve + UVEL[n][loop];
                    TTAve = TTAve + TT[n][loop];
                    DenAve = DenAve + density[n][loop];
                }
            }
        }
        fprintf(fpraw_v, "%f\t", VVelAve/(XAve * YAve * (argc-1)));
        fprintf(fpraw_u, "%f\t", UVelAve/(XAve * YAve * (argc-1)));
        fprintf(fpraw_t, "%f\t", TTAve/(XAve * YAve * (argc-1)));
        fprintf(fpraw_den, "%g\t", DenAve/(XAve * YAve * (argc-1)));
    }
}

```

```
    }  
    fprintf(fpraw_v, "\n");  
    fprintf(fpraw_u, "\n");  
    fprintf(fpraw_t, "\n");  
    fprintf(fpraw_den, "\n");  
}  
fclose(fpraw_u);  
fclose(fpraw_v);  
fclose(fpraw_t);  
fclose(fpraw_den);  
}
```