The Dissertation Committee for Franklin Ted Willmore certifies that this is the approved version of the following dissertation:

Empty Space
and
How Things Move Around in It

Committee:

Isaac C Sanchez, Supervisor

Tom Truskett

Venkat Ganesan

Benny Freeman

Peter J Rossky

Empty Space
and
How Things Move Around in It

by

Franklin Ted Willmore, B.S.; M.S.

Dissertation

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Doctor of Philosophy

The University of Texas at Austin

December 2006

Dedication

Dedicated to my parents, Floyd and Dawn Willmore, for raising me right.

Acknowledgements

I would like to acknowledge Dr Isaac Sanchez for his support and discretion in allowing me the opportunity to pursue graduate study at the University of Texas at Austin.

Empty Space
and
How Things Move Around in It

Publication No._____

Franklin Ted Willmore, PhD

The University of Texas at Austin, 2006

Supervisor: Isaac C Sanchez

All matter contains pockets of empty space. Solubility and transport properties of polymers and other amorphous materials are highly dependent on the character of this free volume. Surprisingly little has been done to characterize these void spaces. Previous research has characterized free volume in terms of geometric tessellations, and spherical cavities, and calculates distributions of the sizes of these cavities. Two polymers might have the same overall free volume, yet exhibit vastly different free volume distributions, with accompanying differences in solubility and diffusivity relative to a particular penetrating species. These differences are due in part to differences in the size distributions of cavities, but the connectivity of the voids also plays an important role. Current free volume models are extended, and new multiscale probability models are introduced.

Table of Contents

List of Tables xii

List of Figures xiii

CHAPTER 1: INTRODUCTION **1**

CHAPTER 2: PRINCIPLES AND METHODOLOGY **5**

The fundamental posulate of statistical mechanics 5

Entropy 5

The Boltzmann Factor 6

Ensembles 7

NVE (Microcanonical) Ensemble 7

NVT (Canonical) Ensemble 8

Monte Carlo Methods 8

Monte Carlo Integration 8

Metropolis Sampling 9

Widom Insertions 9

Molecular Dynamics Simulations 11

Cavity Sizing 11

Probability and Probability Models 13

Conditional Probability 13

Conditional Expectation 13

Bayes' Formula 13

Models of Fluids 15

Hard Spheres 15

Lennard-Jones Fluid 15

Water 16

CHAPTER 3: FREE VOLUME PROPERTIES OF MODEL FLUIDS AND POLYMERS: SHAPE AND CONNECTIVITY 17

Abstract 17

Introduction 18

Experimental 19

Generating polymer configurations 19

The shape parameters 21

Results and discussion 22

Hard spheresand lennard-jones fluid 22

High free volume polymers 23

Insertion-continuum method (ICM) 23

Measuring fractional cavity volume 24

Conclusions 25

CHAPTER 4: POLYMER AGING EFFECTS ON FREE VOLUME 33

Introduction 33

Model and Simulation Details 33

Building Amorphous Cells 33

Simulation of Cavity Size Distribution 34

Simulation of Connectivity and Shape of the Nanopores[58] 35

Connectivity — Span and Radius of Gyration 36

Shape — Nanopore Volume and Surface Area 37

CHAPTER 5: PROBABILISTIC MOLECULAR DYNAMICS: SELF-DIFFUSION IN A LENNARD-JONES FLUID 40

Abstract 40

Introduction 40

The Technique 41

Simulation Details 44

Results and Discussion 45

CHAPTER 6: SMALL MOLECULE DIFFUSION BY PROBABILISTIC MOLECULAR DYNAMICS: MACROSCOPIC DIFFUSIVITY FROM MICROSCOPIC DATA USING A MESOSCALE MODEL 50

Abstract 50

Introduction 50

Part I: Conduction Averaging 52

Site Approximation 56

Pair Approximation 57

Mean-Field Treatment 59

Limiting Probability: 60

Simulation Details 61

Test Cases 63

Part II: Probabilistic Molecular Dynamics 67

Results and Discussion 69

Conclusions 78

CHAPTER 7: MISCELLANY **79**

Free Volume Pair Correlation Function 79

aspherical Parameter of Cavity Clusters 81

CHAPTER 8: RECOMMENDATIONS OF FUTURE WORK **82**

PMD of Diatomics 82

Chemical Potential via PMD 83

APPENDIX A – SOURCE CODE/LIBRARY FUNCTIONS **84**

ftw_std.h 84

ftw_std.c 84

ftw_param.h 84

ftw_param.c 85

APPENDIX B – FILE FORMATS **87**

APPENDIX C – SOURCE CODE/LENNARD-JONES FLUID IMPLEMENTATION **87**

ljx_main.c 87

ljx_main.h 90

io_setup.h 90

io_setup.c 90

graphics.h 93

graphics.c 93

command_line_parser.c 97

energy.h 98

energy.c 98

APPENDIX D – SOURCE CODE/PMD IMPLEMENTATION **100**

utrwt69.c 100

APPENDIX E – SOURCE CODE/CONDUCTION AVERAGE IMPLEMENTATION **109**

matrix.c 109

APPENDIX F – SOURCE CODE/LENNARD-JONES FLUID IMPLEMENTATION **111**

hs_main.h 111

hs_main.c 111

command_line_parser.h 114

command_line_parser.c 114

io_setup.h 116

io_setup.c 116

graphics.h 122

graphics.c 122

REFERENCES **126**

Vita 131

LIST OF TABLES

Table 3-1: Structure and free-volume properties. 32
Table 4-1: Comparison of Average Span Size, Average Radius of Gyration Size, Average Nanopore Volume and Nanopore Surface Area in As-Cast and Aged PTMSP 39
Table 5-1: Description of the rules used to generate the various curves. 44
Table 6-1: Mass and charge transfer analogous quantities and phenomenological ldaws. 53
Table 6-2: Experimental details for simulations. 72
Table 6-3: Diffusivity results (cm^2/s) of tested penetrant/matrix combinations. Both linear and conduction averages over configurations are shown. Experimental values are from Thran, et al[6]. 73

LIST OF FIGURES

Figure 3-1: A set of one or more contiguous overlapping cavities forms a cavity cluster. 27
Figure 3-2: Cluster volume is determined by Monte Carlo integration. Volume is the ratio of points inside one or more cavities to the total number of points sampled, times box size. 27
Figure 3-3: Free volume cluster properties for hard sphere (= 0.80) and Lennard-Jones fluids (T* = 1.2, = 0.80). Average values of shape parameters are shown in the legends. 28
Figure 3-4: Free volume cluster properties for TFE-BDD copolymer and PTMSP. Both are at a density of 0.75 g/cm^3 Average values of shape parameters are shown in the legends. 29
Figure 3-5: Volume-weighted cavity size distributions (without clustering) for TFE-BDD and PTMSP. 30
Figure 3-6: Fractional free volume as a function of probe size using ICM compared with Bondi free volume. 30
Figure 3-7: Graphical representation of fractional free volume in a sample of TFE-BDD shown for Xenon alone shown below, and for Helium (red), Neon (orange), Argon (yellow) and Xenon (green), shown above. 31
Figure 4-1. Comparison of nanopore span distributions in as-cast (ρ = 0.75g/cm^3) and aged (ρ = 0.85 g/cm^3 and ρ = 0.95 g/cm^3) PTMSP. 38
Figure 4-2. Comparison of nanopore radius of gyration distributions in as-cast (ρ = 0.75g/cm^3) and aged (ρ = 0.85 g/cm^3 and ρ = 0.95 g/cm^3) PTMSP. 38
Figure 4-3. Comparison of nanopore volume distributions in as-cast (ρ = 0.75g/cm^3) and aged (ρ = 0.85 g/cm^3 and ρ = 0.95 g/cm^3) PTMSP. 39
Figure 4-4. Comparison of nanopore surface area distributions in as-cast (ρ = 0.75g/cm^3) and aged (ρ = 0.85 g/cm^3 and ρ = 0.95 g/cm^3) PTMSP 39
Figure 5-1: Running average of diffusivity obtained for density 0.1, T*=6.0 vs time in picoseconds. 46
Figure 5-2: The BF and 2P curves give the best results at this temperature, with the 2P being closest to the published molecular dynamics values 47
Figure 5-3: As temperature increases, the monte carlo results increasingly underpredict the molecular dynamics results. 48
Figure 5-4: At the highest reduced temperature studied, the monte carlo results continue the trend of increasingly underpredicting the self-diffusivity. The enhancement of adding the random uncertainty term in the 2P-0_2 and 2P-1_0 series offsets this only slightly at low densities where the result is most off, and actually overshoots the molecular dynamics results at higher densities. 49
Figure 6-1. Conduction averaging uses an infinite grid of containers and hopping probabilities to determine an effective conductivity for a composite material as if it were made of these individual cells. 62
Figure 6-2: The conduction average is consistently higher value than linear and reciprocal averages, because it allows for transport around more insulating elements, as occurs in real materials. Shown is an equal mixture of two types of elements, one with D = 1, the other with D = D_{max}. 64
Figure 6-3: Resulting effective diffusivity from a mixture of elements of D = 1, with elements of type D = 2 or D = 4, as shown. 65

Figure 6-4: Effect of placing an single element of higher diffusivity into a matrix of with all other elements having diffusivity D = 1. The effectivity diffusivity D_{eff} asymptotes to a value of 3. The element occurs with frequency 1ppm, using 10000 samples runs, each for 1000 steps.. 66
Figure 6-5: Markers show mean-square displacement vs time for He in polystyrene via PMD. Average is over 10 configurations. Straight line is fit to data from 200-1000ps..... 74
Figure 6-6: Comparative time series for traditional and probabilistic molecular dynamics for Ne in PSF-M..... 75
Figure 6-7: Diffusivities obtained by traditional vs probabilistic molecular dynamics in cm2/s. The two outlier points are for methane diffusion in 6FDA-6FpDA and 6FDA-6FmDA. By recomputing the average over configurations for MD using conduction averaging, the values come in line with the PMD values..... 76
Figure 6-8: Diffusivity values in cm^2/s from PMD vs published experimental values. The two outlier points are for methane diffusion in 6FDA-6FpDA and 6FDA-6FmDA. This is attributed to an inadequacy of the forcefield model for describing the interactions with these highly fluorinated compounds, also observed with traditional molecular dynamics..... 77
Figure 7-1: Free volume pair correlation function..... 80

CHAPTER 1: INTRODUCTION

This work began with the search for a way to characterize the nanoscopic connectivity of void spaces. It ends with a recipe called probabilistic molecular dynamics[1] for calculating diffusion coefficients *via* a technique which approximates the motion of small molecules through these void spaces. There were many detours along the way. The cavity overlap analysis technique[2] was developed and applied to a simulated polymer aging process[3]. An alternative[2] to the Bondi[4] method for measuring fractional free volume was conceived. A free-volume pair correlation function was developed; it was interesting, but we never did anything with it. We attempted to apply our 'probabilistic molecular dynamics' to self-diffusion in a Lennard-Jones fluid, but were not particularly happy with the results. Many ideas were conceived, several were investigated, and three papers were published.

Both thermodynamic and kinetic properties of materials are closely related to the structure of free volume. Free volume theory[5, 6] predicts a linear relationship between the logarithm of the diffusivity and the reciprocal of the fractional free volume. A similar relationship has been shown to exist between the fractional cavity volume[7-9] and diffusivity. Solubility of a gas (a thermodynamic property) and corresponding Henry's law behavior can be predicted by Widom insertions[10].

Experimentally, these nanospace properties are evaluated by means of Positron Annihilation Lifetime Spectroscopy (PALS)[11-19], mechanical measurements of equation of state behavior[20], and recently by Xe-NMR[21]. Theoretical and simulation-based approaches include Voronoi tessellations[14, 22-27] and the energy-based Cavity Energetic Sizing Algorithm (CESA) for sizing cavities in liquids[7]. The recently introduced EVEBAT[28] uses force-fields as its basis for

defining free volume. The CESA method was recently applied to high-free volume polymers, the results[8] illustrating good agreement with PALS data for the same materials. Differences observed between TFE/BDD and PTMSP, which have a similar Bondi free volume but markedly different diffusivities with respect to CO_2 are explained by larger cavity sizes in PTMSP[8]. It is generally accepted that diffusion occurs mechanistically as a series of 'hops' between cavities which change shape and become periodically connected or disconnected, allowing the penetrant to escape one void and travel to another[29-34]. Transition state theory has been applied to this hopping mechanism to attempt to understand the kinetic behavior[29-32, 35]. Recent research has focused on bridging the understanding between cavity sizing using PALS and theoretical cavity sizing methods[36-39]. The addition of nanoparticles has been used to alter the way polymer chains pack and thus change the average cavity size and resulting diffusion properties[11]. The effect of encaged aromatic guests on free volume properties has also been considered[40]. Special ladder-type polymers of intrinsic mobility (PIM's) have been engineered for special applications[41]. The connectivity of the void space is of interest because it holds the key to the mechanism of transport through the material.

The effects of aging on the free volume distribution in a set of high free volume polymers were studied using the cavity energetic sizing algorithm(CESA)[7, 8] and the cavity overlap technique[2]. Rapid physical aging of Poly(1-trimethylsilyl-1-propyne) (PTMSP), the most permeable polymer known, is characterized by an increase in bulk density and a reduction in observed cavity sizes. Observed experimentally by PALS and theoretically by CESA, this reduction in observed cavity sizes is accompanied by a decrease in the small molecule diffusivity and a reduction in the permeability. The trend is for the larger cavities and also the larger spans of connectivity to disappear with aging. It is unclear whether the actual mechanism of this aging

process involves the sudden collapse of the larger holes, the decay of larger holes into smaller holes, or an overall contraction of the material where all holes tend to shrink proportionally as the system ages.

Molecular dynamics reflects a comprehensive understanding of all that is known about the motions and behaviors of atoms and molecules from an ab-initio standpoint. It is elegant and physical, detailed and thorough. Sok et al[42] have effectively demonstrated the strength of molecular dynamics for predicting the diffusivity of small molecules in a polymer, with results that strongly reflect the importance of the motion of the polymer matrix, as it opens, closes, and reshapes voids which allow penetrant to travel. Tsige and Crest[43, 44] have used bead-spring models of polymers to look at the motion of solvent using molecular dynamics, and have observed Fickian diffusion over a range of diffusivities. Their model also accounts for motion of the matrix. Coarse-graining and the approximation of distributions with appropriately sampled values as inputs to a stochastic model has found wide application. The 'equation-free' approach of Rico-Martinez et al.[45] uses a coarse-grain projective stochastic model to reproduce the limit cycle behavior of a monomer-dimer-poison surface model[46] of Vigil et al. Gauthier and Slater have examined diffusion through a heterogeneous lattice by impleme nting blocking sites and external fields[47-49]. Armatas, et al[50] have used a Monte Carlo scheme in populating a network with pores of different types to obtain an effective pore diffusivity for the composite material[50], yielding normal Fickian diffusion. Zielinski and Duda[51] developed a model to predict small molecule diffusivity without experimental data, based on the joint probability of finding a sufficiently large void into which to move and the possibility of having the energy to escape from the current position. Their model produced surprisingly good results considering that it does not take into account the specifics of hole morphology and connectivity for particular

polymers. Later studies consider the role of thermal fluctuations[52] and local geometry[53, 54]. Han and Boyd[55] look at methane in polystyrene as a series of hops from pore to pore, using jump lengths and probabilities to characterize the diffusion mechanism.

CHAPTER 2: PRINCIPLES AND METHODOLOGY

THE FUNDAMENTAL POSULATE OF STATISTICAL MECHANICS

The fundamental postulate or *equal a priori probability postulate* in statistical mechanics may be stated as follows[56]:

Given an isolated system in equilibrium, it is found with equal probability in each of its accessible microstates.

That a system does not prefer any one of its available microstates over any other is a clarification of the general notion that a system will prefer to be in a 'lower energy' state. It is more accurate to express that there are simply more states available at a lower energy. In a system with Ω given microstates at a particular energy, the probability of finding the system in any particular microstate is $p = 1/\Omega$.

This postulate is necessary because it allows one to conclude that for a system at equilibrium, the thermodynamic state (macrostate) which could result from the largest number of microstates is also the most probable macrostate of the system.

ENTROPY

What ties together the ideas of probability and energy is the concept of entropy. Often misunderstood or vaguely described as the 'degree of randomness' of a system, entropy is more fundamental and is best described as a measure of the number of possibilities. A classic example is the number of ways N molecules of gas may be distributed between two identical and non-reactive containers in equilibrium. If there is only molecule, it will be found with equal probability in either container. If there are two, there are four possibilities: Both are in the first container, both are in the second, or one in each (two possibilities). As more molecules are

added, it can be shown that there are more possible arrangements when the molecules are divided nearly equally between the two containers, and that the most probable arrangement is that they are divided exactly equally. The impetus of these non-interacting molecules towards distributing equally between the two containers can be described as a pure depletion force; there is no repulsion to drive them away from another, simply more possibilities, more entropy, more ways to order themselves, and a resultant lower free energy to be achieved in so doing.

This allows for the definition of the *information function* (in the context of information theory):

$$I = \sum_i \rho_i \ln \rho_i = < \ln \rho_i >$$

Generalizing for a number of such identical containers, we can express the number of possible arrangements in terms of the fraction of molecules ρ_i in each container. When all of the fractions in all containers are equal, I is minimal, which reflects the fact that we have minimal information about the system. When our information is maximal, i.e. one ρ_i is equal to unity and the rest are equal to zero (we know exactly what state our system is in), the function I is maximal. This "information function" is the same as the reduced entropic function in thermodynamics. Saying that maximum entropy is favored is equivalent to saying that the system is most likely to be found with equal fractions of molecules distributed among all containers.

THE BOLTZMANN FACTOR

The probability p_j that a system will be in a state with energy E_j depends exponentially on the energy of that state, or

$$p_j \propto e^{-E_j / k_B T}$$

The sum of probabilities must equal one, thus the normalization constant will be 1/Q, where Q is called the partition function, or 'sum over states':

$$Q = \sum_j e^{-E_j / k_B T}$$

This can be viewed as stating that the probability of a system being in a state with energy E_{j+1} is proportional to the probability of it being in state with energy E_j, tantamount to a 'linear attenuation' of probabilities as energy increases, thus giving the decaying exponential character of the Boltzmann distribution.

Recently, an interesting result has been obtained based on this distribution which allows equilibrium free-energy differences to be obtained from non-equilibrium measurements[57, 58].

ENSEMBLES

NVE (MICROCANONICAL) ENSEMBLE

The molecular dynamics simulations required for this study were performed using the microcanonical or constant NVE ensemble. The temperature in such a simulation tends to fluctuate about the equilibrium temperature but the overall energy of the system is constant.

The second law of thermodynamics applies to isolated systems. The Microcanonical ensemble describes an isolated system. The entropy of such a system can only increase, thus the maximum of its entropy corresponds to an equilibrium state for the system. Because an isolated system keeps a constant energy, the total energy of the system does not fluctuate. Thus, the system can access only those of its micro-states that correspond to a given value *E* of the energy. The internal energy of the system is then strictly equal to its energy.

We define $\Omega(E)$ as the number of micro-states corresponding to this value of the system's energy E. The macroscopic state of maximal entropy for the system is the one in which all micro-states are equally likely to occur during the system's fluctuations.

NVT (CANONICAL) ENSEMBLE

The simulations of water, hard-sphere and Lennard-Jones fluids in this work were performed using the canonical or constant NVT ensemble. In the canonical ensemble, the probability P_i that a macroscopic system in thermal equilibrium with its environment will be in a given microstate with energy E_i is

$$P_i = \frac{e^{-\beta E_i}}{Q}$$

where $\beta = \frac{1}{kT}$, $Q = \sum_j \exp(-\beta E_j)$ is the canonical partition function, and E_i is the energy of the i^{th} microstate of the system. The partition function is a measure of the number of states accessible to the system at a given temperature. The probabilities of the various microstates must add to one, and the partition function is the normalization factor. In such a system, the Helmholtz free energy A is directly related to this partition function as:

$$A = -kT \ln Q$$

MONTE CARLO METHODS

MONTE CARLO INTEGRATION

Monte Carlo integration can be performed by comparing a complicated system of interest to a more well known system. For example, the volume of an irregularly shaped object may be determined by such an integration by placing the irregularly shaped object into a larger container whose volume is already known. By picking points at random within the container, the volume

of the irregularly shaped object can be determined as the fraction of sample points which lie inside of the object times the volume of the container.

METROPOLIS SAMPLING

The Metropolis[59] sampling algorithm used in Monte Carlo simulation operates by sequential attempts to move molecules in the system in random, unphysical movements. The moves are accepted or rejected based on the system temperature and the energy change associated with the move. The only move allowed in the NVT ensemble is a change in position and is accepted with the probability $\exp(-\Delta E/kT)$ when the energy change ΔE is positive, or unconditionally if the energy change is negative.

WIDOM INSERTIONS

Chemical potential of a species alone or in a mixture can be shown to be a function of insertion energy[60]. The canonical partition function for a system of N identical, indistinguishable molecules in a fixed volume V in equilibrium with a heat bath at temperature T is written as:

$$Q(N,V,T) = \frac{V^N}{\Lambda^{3N} N!} \iint \ldots \int d\vec{r}^{\,N} \exp\left\{-\beta U\left(\vec{r}^{\,N}\right)\right\}$$

The Helmholtz free energy for this system can be expressed as:

$$F = -kT \ln Q$$

This can be expressed as the sum of a contribution from ideal behavior and a contribution for the interaction of the particles by way of a potential energy function U:

$$F = -kT \ln\left(\frac{V^N}{\Lambda^{3N} N!}\right) - kT \ln \iint \ldots \int \exp\{-\beta U(\vec{r}^{\,N})\}$$

$$F = F_{ig} - F_{ex}$$

From thermodynamics, we know that the chemical potential for this system can be expressed as a derivative of the Helmholtz energy with respect to number of molecules:

$$\mu = \left(\frac{\partial F}{\partial N}\right)_{T,V} = -kT \ln \frac{Q(N+1,V,T)}{Q(N,V,T)}$$

Substituting the expression for the partition function gives:

$$\mu = -kT \ln \frac{V^{N+1}\Lambda^{3N} N!}{V^N \Lambda^{3N+3}(N+1)!} - kT \ln \frac{\iint \ldots \int \exp\{-\beta U(\vec{r}^{\,N+1})\} d\vec{r}^{\,N+1}}{\iint \ldots \int \exp\{-\beta U(\vec{r}^{\,N})\} d\vec{r}^{\,N}}$$

Simplifying, gives:

$$\mu = -kT \ln \frac{V}{\Lambda^3 (N+1)} - kT \ln \int \exp\{-\beta[U(\vec{r}^{\,N+1}) - U(\vec{r}^{\,N})]\} d\vec{r}^{\,N+1}$$

The chemical potential like the Helmholtz free energy can be viewed as the sum of an ideal gas contribution plus an excess chemical potential:

$$\mu = \mu_{ig} + \mu_{ex}$$

This results in an expression for the excess chemical potential in terms of an insertion factor B, which can be valuated as an integral over all possible positions and orientations for the insertion of the N+1th molecule:

$$\mu_{ex} = -kT \ln B, B = \int_{\vec{r}^{\,N+1}} \exp\{-\beta \Delta U(\vec{r}^{\,N} \rightarrow \vec{r}^{\,N+1})\} d\vec{r}^{\,N+1}$$

This integral is generally evaluated by Monte Carlo integration, i.e. a test particle (the $N+1^{th}$ particle) is inserted at various positions and in various orientations into the system, the change in the potential function is evaluated. Samples are repeated until a statistically satisfactory average value of the insertion factor B is obtained.

MOLECULAR DYNAMICS SIMULATIONS

The idea of molecular dynamics is to perform a numerical integration of Newton's equations of motion to determine the positions and velocities of all particles in the system as a function of time. It can be used to model both equilibrium and non-equilibrium dynamical systems. Integration time steps are typically on the order of a femtosecond (10^{-15} seconds). Time steps can be increased or decreased to allow for faster simulations or for more accuracy, respectively. Traditional molecular dynamics techniques typically run on the order of nanoseconds (1ns = 10^{-9} seconds). After a few picoseconds (10^{-12} seconds) of simulation time, it is typical for round-off errors to cause a drift of the temperature scale which must be adjusted periodically.

CAVITY SIZING

The CESA[7] method was used to locate and size cavities and is described briefly here:

1. Assign an energy equivalent to the repulsive part of the corresponding Lennard-Jones potential to each molecular center in the simulation box. The superposition of all such potentials will form an energy landscape with respect to the insertion of a test particle.
2. Select a point at random within the simulation box for the insertion of a test particle of zero diameter.

3. Evaluate the energy gradient at this point.
4. Move the test particle a finite distance down the gradient.
5. Repeat steps 3 and 4 until the test particle is at an energetic minimum.
6. Use this location as the center of the cavity/test particle.
7. Assign an energy equivalent to the attractive part of the corresponding Lennard-Jones potential to each molecular center in the simulation box.
8. Increase the size of the test particle until a diameter is reached where the repulsive and attractive parts of the potential are equal.
9. This diameter is the cavity diameter.

Redundant cavities (duplicate cavities with the same center) are discarded, as are cavities with erroneous diameters, i.e. the attractive and repulsive parts of the energy relationship only sum to zero for a negative or complex value of the diameter. In this way, an exhaustive list of all of the cavities in the system is generated. By considering the overlap of spherical cavities obtained using CESA, we work to understand the connectivity of the void spaces. Two cavities are considered overlapping if the distance between their centers is less than the sum of their diameters. Overlapping cavities are grouped into a new entity, called a cluster (figure 1). Each cavity belongs to only one cluster, and each cluster contains at least one cavity. The shapes of these cluster objects is then characterized according to a set of 'shape parameters': Volume, surface area, span, and radius of gyration.

PROBABILITY AND PROBABILITY MODELS

CONDITIONAL PROBABILITY

The conditional probability that event E occurs, given that event F occurs, is:

$$P(E \mid F) = \frac{P(EF)}{P(F)}$$

where P(EF) represents the event that E *and* F occur, i.e. an event occurs that is in the intersection of the events in E and the events in F.

CONDITIONAL EXPECTATION

The expectation value for the random variable X, given that random variable Y has the value *y*, is:

$$E[X \mid Y = y] = \sum_x xP\{X = x \mid Y = y\} \text{ for discrete } x$$

$$E[X \mid Y = y] = \sum_x xP\{X = x \mid Y = y\} \text{ for continuous } x$$

BAYES' FORMULA

For events E and F, we can express E as:

$$E = EF \cup EF^C$$

Where F^C (read F-complement) is the set of all events not included in F. Since F and F^C are mutually exclusive, we can write:

$$P(E) = P(EF) + P(EF^C)$$

$$P(E) = P(E \mid F)P(F) + P(E \mid F^C)P(F^C)$$

$$P(E) = P(E \mid F)P(F) + P(E \mid F^C)(1 - P(F))$$

This formula proves useful by providing a formal algebra to use when analyzing events with simultaneous characteristics.

MODELS OF FLUIDS

HARD SPHERES

Hard spheres are a fabulous first-approximation to molecules. They are elegant and efficient and reproduce a surprising amount of physicality of real molecules, including a solid-gas phase transition.

LENNARD-JONES FLUID

The Lennard-Jones fluid, based on a Lennard-Jones potential[61], is a simple model for describing phase behavior. It is elegant, yet produces the three most common phases of matter: solid, liquid, and gas, and does so quite remarkably for simple fluids such as noble gases or methane, and with only two parameters describing the particular fluid. The potential form consists of an attractive and a repulsive term, often (but not always) taken to be functions of the reciprocal of the separation distance to the sixth and twelfth powers, respectively:

$$\phi_{ij} = 4\varepsilon\left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6}\right]$$

This form is referred to as a Lennard-Jones 6-12 potential. With the repulsive exponent to the 9th power, this is referred to as a Lennard-Jones 6-9 potential. Both forms are used in this work. The functional form is similar in appearance to a Morse potential, but as a polynomial rather than exponential function, it is computationally much faster to evaluate, and lends itself much more readily to brute-force computation.

WATER

Currently, the most popular models of water are the TIPS models[62-64] and the SPC[65-67] models water. In the simulations performed in this work, the SPC/E water model was used, as implemented by the popular Materials Studio[68] software package. This model consists of a Lennard-Jones 6-12 potential centered on the oxygen molecule, along with partial electrical charges of +0.41 and -0.82 overlaid on each of the hydrogens and oxygen, respectively.

CHAPTER 3: FREE VOLUME PROPERTIES OF MODEL FLUIDS AND POLYMERS: SHAPE AND CONNECTIVITY

(Published in J Polymer Science B)[2]

ABSTRACT

The Cavity Energetic Sizing Algorithm (CESA) method of int Veld [7] is extended to characterize the nonspherical nature of free volume. The new technique is introduced with reference to simple model fluids (water, hard spheres, and a Lennard-Jones fluid) and then applied to polymers of interest to membrane scientists. A set of shape parameters is introduced, characterizing nanopores in terms of surface area, volume, radius of gyration, and span. Results are presented for a Lennard-Jones fluid and a hard sphere fluid, and for the high free volume polymers (poly-trimethyl-silyl-propane) poly(1-trimethylsilyl-1-propyne) (PTMSP) and a random copolymer of 2,2-bis(trifluoromethyl)-4,5-difluoro-1,3-dioxole (TFE/BDD). PTMSP is observed to have an average free volume cluster span of 1.43 nm, compared to TFE/BDD with an average cluster span of 0.98 nm, consistent with the markedly higher permeability of CO_2 observed in PTMSP. An additional method for measuring free volume is introduced, similar to a method introduced by Greenfield and Theodorou[69-72], which measures free volume relative to a specific probe. The method captures 1-3 times the fractional cavity volume captured by CESA. Free volume measurements are presented for a set of polysulfones with respect to noble gas probes.[11, 73]

INTRODUCTION

The solubility and diffusivity of a penetrant species in an amorphous material are determined by the nanoscale properties of the free volume of the material. Experimentally, these nanospace properties are evaluated by means of Positron Annihilation Lifetime Spectroscopy (PALS)[11-19], mechanical measurements of equation of state behavior[20], and recently by Xe-NMR[21]. Theoretical and simulation-based approaches include Voronoi tessellations[14, 22-27] and the energy-based Cavity Energetic Sizing Algorithm (CESA) for sizing cavities in liquids[7]. The recently introduced EVEBAT[28] uses force-fields as its basis for defining free volume. The CESA method was recently applied to high-free volume polymers with good results[8]. The technique captures 25-50% of the Bondi free volume of the system, and provides a distribution of cavity sizes, which are correlated to permeability properties. Differences observed in CO_2 diffusivities between TFE/BDD and PTMSP, which have a similar Bondi free volume but markedly different with respect to CO_2 are explained by larger cavity sizes in PTMSP[8]. It is generally accepted that diffusion occurs mechanistically as a series of 'hops' between cavities which change shape and become periodically connected or disconnected, allowing the penetrant to escape one void and travel to another[29-34]. Transition state theory has been applied to this hopping mechanism to attempt to understand the kinetic behavior[29-32, 35]. Recent research has focused on bridging the understanding between cavity sizing using PALS and theoretical cavity sizing methods[36-39]. The addition of nanoparticles has been used to alter the way polymer chains pack and thus change the average cavity size and resulting diffusion properties[11]. The effect of encaged aromatic guests on free volume properties has also been considered[40]. Special ladder-type polymers of intrinsic mobility (PIM's) have been engineered for special applications[41]. The

connectivity of the void space is of interest because it holds the key to the mechanism of transport through the material.

EXPERIMENTAL

The model hard sphere fluid and Lennard-Jones fluid were simulated at the same density, using the hard sphere diameter and the Lennard-Jones sigma parameter as the reference length scale when comparing results between the two. The Lennard-Jones fluid uses a truncated and shifted potential of the form:

$$\phi_{ij} = 4\varepsilon\left(\left(\frac{\sigma}{r}\right)^{6} - \left(\frac{\sigma}{r}\right)^{12}\right) - \phi_{r=2.5}, 0 \le r < 2.5$$

$$\phi_{ij} = 0, 2.5 \le r$$

For both the hard sphere and the Lennard-Jones simulations, 25 configurations of 1000 atoms were used. The initial condition for the hard spheres was an FCC lattice, scaled to the size of the box. The initial condition for the LJ fluid was random placement of each center in the simulation box. In each case, the system was allowed to run for 10000 Monte Carlo cycles before the first sample configuration was recorded. Subsequent configurations were recorded every 5000 Monte Carlo cycles.

GENERATING POLYMER CONFIGURATIONS

The materials of interest were were generated by molecular dynamics using the the amorphous cell module of the commercially available Accelrys Materials Studio[68] software package. The COMPASS force-field (Condensed-phase Optimized Molecular Potentials for Atomistic Simulation Studies) was used in all simulations. A cubic simulation box with periodic boundary conditions was used. For PTMSP, the polymer chain is 50 repeat units, with random

choice of cis/trans monomer units. Two such chains were folded into the amorphous cell/simulation box for each configuration, for an experimental density of 0.75 g/cm^3 and a cell dimension of 2.92 nm. Sixty initial configurations were produced and then system energy minimized for 5000 steps for each configuration. Next, each configuration was run for 10ps of NVT molecular dynamics at 298 K. For TFE/BDD, polymer chains of 100 repeat units (random copolymer with 13% chance of TFE and 87% chance of BDD at each unit) were constructed at a density of 1.74 g/cm^3. The sizes of the resulting simulation boxes were approximately 2.8nm. Sixty such initial states were created and energy was minimized for 10000 steps, followed by an NVT MD run of 10ps at 298K.

The CESA method was used to locate and size cavities and is described briefly here:

Assign an energy equivalent to the repulsive part of the corresponding Lennard-Jones potential to each molecular center in the simulation box. The superposition of all such potentials will form an energy landscape with respect to the insertion of a test particle.

Select a point at random within the simulation box for the insertion of a test particle of zero diameter.

Evaluate the energy gradient at this point.

Move the test particle a finite distance down the gradient.

Repeat steps 3 and 4 until the test particle is at an energetic minimum.

Use this location as the center of the cavity/test particle.

Assign an energy equivalent to the attractive part of the corresponding Lennard-Jones potential to each molecular center in the simulation box.

Increase the size of the test particle until a diameter is reached where the repulsive and attractive parts of the potential are equal.

This diameter is the cavity diameter.

Redundant cavities (duplicate cavities with the same center) are discarded, as are cavities with erroneous diameters, i.e. the attractive and repulsive parts of the energy relationship only sum to zero for a negative or complex value of the diameter. In this way, an exhaustive list of all of the cavities in the system is generated. By considering the overlap of spherical cavities obtained using CESA, we work to understand the connectivity of the void spaces. Two cavities are considered overlapping if the distance between their centers is less than the sum of their diameters. Overlapping cavities are grouped into a new entity, called a cluster (figure 3-1). Each cavity belongs to only one cluster, and each cluster contains at least one cavity. The shapes of these cluster objects is then characterized according to a set of 'shape parameters': Volume, surface area, span, and radius of gyration.

THE SHAPE PARAMETERS

Volume: Volume of the nanopore is calculated by performing a straightforward Monte Carlo integration (figure 3-2). Points are selected at random from within the simulation box, and are determined to lie within at least one of the component cavities of the cluster or not. This assures that the volume of overlapping regions is not over-counted. The cluster volume obtained is the ratio of points inside the cluster to the total number of points selected, times the box volume.

Surface Area: Cluster surface area is determined by selecting points that lie on the surface of each component cavity within the cluster. Sets of zenith and azimuth (longitude and latitude) are selected randomly to generate points uniformly on the surface of each sphere in the cluster. Zenith angles are sampled from a sinusoidal distribution, in order to prevent divergence in the sampling density at the poles of the sphere. The exposed surface area fraction, ϕ_i, of each

component sphere is calculated by counting the number of points which lie on the surface of each sphere (not inside the radius of any other sphere) and dividing by the total number of points sampled on the sphere. The surface area of the cluster is then determined by multiplying this fraction by the surface area of that sphere σ_i and summing over all spheres. Thus, the cluster surface area is $\sigma_{cluster} = \Sigma_i \phi_i \sigma_i$.

Span: The span is defined as the farthest distance between any two points that lie within the cluster. Analytically, this is calculated by finding the two cavity centers in the cluster with the greatest distance between them. The span is this distance between centers, plus the radius of each of the two farthest cavity centers.

Radius of Gyration: In general, the radius of gyration of an object is defined as the mass averaged root mean square distance from the center of mass. (This differs slightly from the radius of gyration about a specific linear axis, which is used to define the moment of inertia of an object). The radii of gyration of the clusters in this work were calculated by selecting points at random within the cluster and assigning to each of them an equal 'weight'. The center of mass of this set of points was then determined, and the radius of gyration determined as the root mean square distance of the set points in the set from the center of mass of the set. This was performed on a 'per cluster' basis.

RESULTS AND DISCUSSION

HARD SPHERESAND LENNARD-JONES FLUID

Two simple fluids were compared: a hard sphere fluid at a reduced density of 0.80, and a Lennard Jones fluid at a reduced density of 0.80 and reduced temperature of 0.72 (liquid at liquid-vapor coexistence). Hard spheres, with no attractive forces, have the smallest overall

cluster volumes, as the spheres float freely and tend to 'jam up' what would otherwise be connected void space. The Lennard-Jones fluid, with its intermolecular attractions, exhibits a larger overall free space with this method. This free space is also more connected, as illustrated by having larger overall free volume cluster sizes, clusters of larger span, and clusters of larger radii of gyration, as shown in figure 3-3. Note that the distributions generated have been weighted by cluster volume, i.e. selecting a point at random from all of the volume captured by all of the cavitites, what is shown is the probability that it would lie in a cluster with the given value of surface area, volume, etc. Average values of each parameter are shown in the legend. There is a slight discrepancy in the lower limit values of span and radius of gyration arising from data smoothing, when these parameters do in fact approach zero.

HIGH FREE VOLUME POLYMERS

We also applied this technique to the high free volume polymers PTMSP and TFE/BDD. While PTMSP already shows the tendency toward larger spaces of free volume using CESA alone, the trends toward larger span and larger radius of gyration are more pronounced using the new method (figure 3-4). The original volume weighted cavity size distribution is shown in figure 3-5. Since the presence of very large cavities tends to absorb a great deal of the free volume, it is difficult to get good statistics, particularly for the span and radius of gyration at larger sizes. The data shown reflects a fair amount of smoothing and the peaks have no particular significance.

INSERTION-CONTINUUM METHOD (ICM)

CESA as well as the cavity overlap technique based upon it work well to capture the character of free volume, but suffer an unfortunate consequence due to cavity centers being

found in terms of an energetic repulsive minimum: This assures that large amounts of free space are forsaken, in favor of choosing instead a local or not-so-local repulsive minimum at the center of free space. A new technique is introduced here which captures more of the free space.

Like the hard sphere method introduced by Greenfield and Theodorou[69], ICM attempts to develop a silhouette of free volume in terms of test particle insertion points. Since it (like CESA) uses a soft potential, it finds accessible void space that hard spheres would miss. And since it does not force the test particle to an energetic minimum (as does CESA) the amount of accessible free volume captured tends to be larger than the fractional cavity volume captured by CESA. Also, inherent in this model is a more physical explanation for why the captured volume is accessible to the penetrant, something which tessellation methods particularly neglect.

ICM was originally called the 'land and sea' model, in that it separates space into two regions, one where a penetrant is not likely to be found ever (the land) and a region where our penetrant is very likely to be found floating about (the sea) based on the insertion energy for the penetrant into the configuration. On this energy landscape, 'sea level' is an arbitrarily chosen energy level (3/2 kT is used here, the average thermal/kinetic energy of a penetrant species) on the insertion energy landscape which distinguishes what regions are land (energy above) and sea (energy below).The technique is implemented by overlaying the simulation box with a grid and making trial insertions of a test particle at each grid point. Insertion energy is measured at each point; points that lie below the 'sea level' insertion energy are the sea, those that lie above are the land, systematically mapping out the free space with respect to the test particle species.

MEASURING FRACTIONAL CAVITY VOLUME

ICM was used to measure the fractional free volume of our two test polymers. Configurations of each polymer were probed using helium and xenon as penetrants. The volume

measured for PTMSP is in good agreement with that measured by the Bondi method, but TFE/BDD showed a considerably lower 'available' free volume [table 3-1]. The larger cavity sizes which CESA observes in PTMSP are accentuated by ICM as reflected by PTMSP exhibiting nearly twice the free volume of TFE/BDD when using the large Xenon probe.

The method was also applied to a set of polysulfones [figures 3-6 and 3-7] to illustrate the effect of probe size. A smaller probe will obviously fit smaller spaces, and thus capture more free volume, but how much the free volume varies with probe size gives an indicator of how constricted the spaces are. Whereas the high free volume polymers exhibit near or greater the Bondi free volume even with a Xenon probe, the polysulfones barely approach the Bondi volumes even with the tiny Helium probe.

The same procedure used to analyze overlapping cavity clusters can theoretically be applied to the volume contained by insertion points generated using ICM, with the following restrictions: The number of successful insertion points obtained using this method can be quite large, and thus computationally unwieldly. High free volume systems evade analysis as the cluster objects have greater continuity and often percolate. In the limit of shrinking penetrant size or with increasing free volume, all structures percolate. Percolation presents an obvious problem of singular nature when attempting the cluster analysis.

CONCLUSIONS

An algorithm is developed which provides a set of quantitative measures of the connectivity of void space in amorphous materials. Of these measures, the span of overlapping cavities provides the strongest distinction between the polymeric materials studied. Overlapping cavities in PTMSP span nearly twice the average distance as those in TFE/BDD. This greater overlap of cavities suggests a possible mechanism of transport of gas through the material,

explaining the higher observed diffusivity of 3.0 x 10^{-5} cm^2/s for CO_2 in PTMSP vs 0.56 x 10^{-5} cm^2/s in TFE/BDD.

Free volumes predicted using the Bondi method may overpredict the 'available' free volume, the fraction of space where a penetrant particle is likely to be found. Bondi free volumes are macroscopic measures of free volume, whereas CESA and the insertion-continuum method capture free volume as specific locations. CESA captures free volumes on the order of 25-50% of the Bondi free volume. The insertion-continuum method quantitative captures approximately 25%-100% of the Bondi free volume depending on which probe is used. Less free volume is captured for TFE/BDD than for PTMSP when using the larger xenon probe [table 3-1]. This suggests more constricted free volume in the TFE/BDD, consistent with the lower diffusivity observed for CO_2.

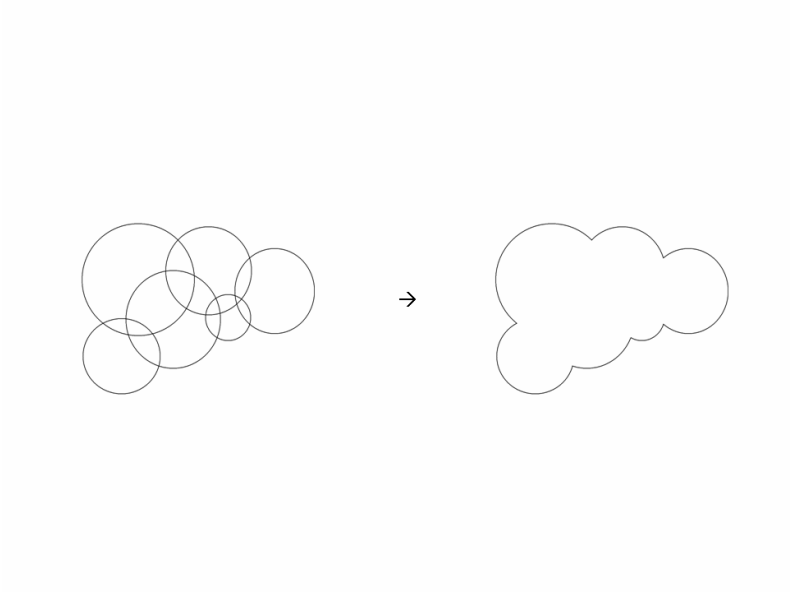


Figure 3-1: A set of one or more contiguous overlapping cavities forms a cavity cluster.

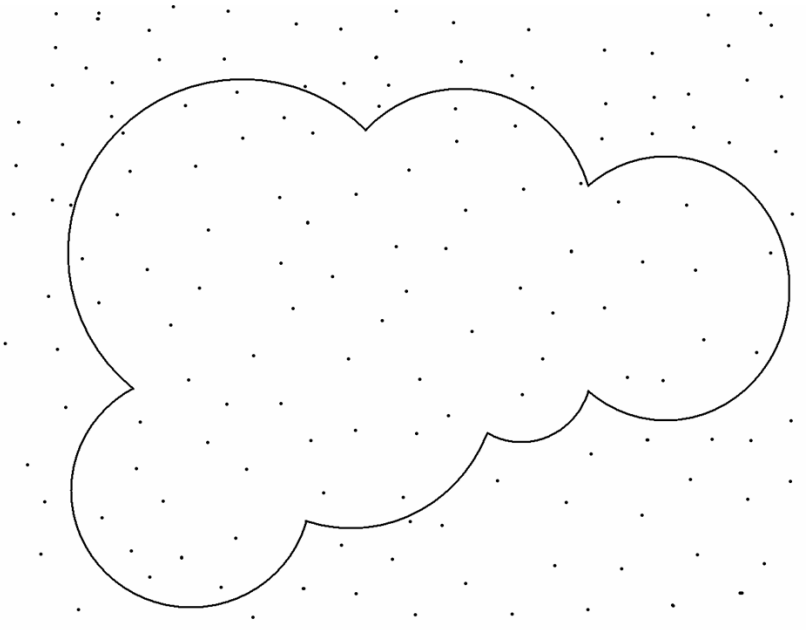


Figure 3-2: Cluster volume is determined by Monte Carlo integration. Volume is the ratio of points inside one or more cavities to the total number of points sampled, times box size.

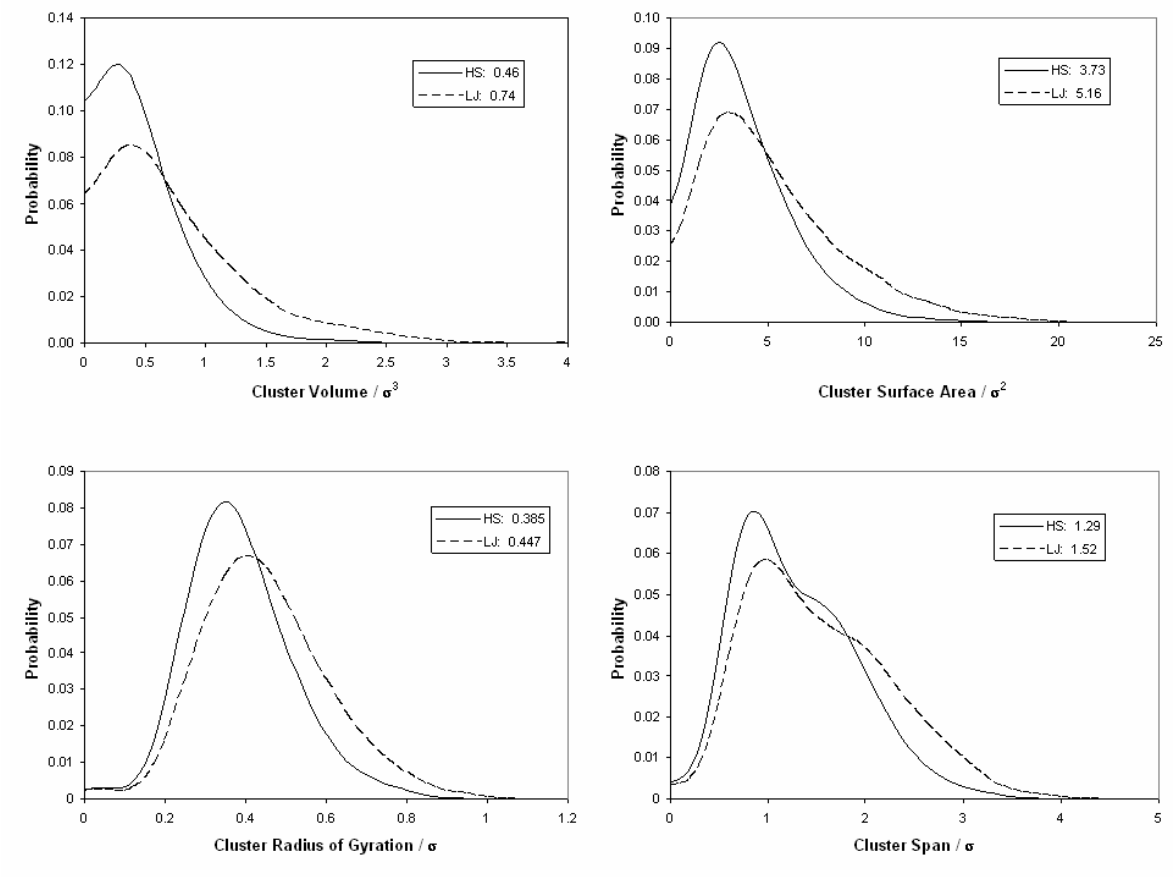


Figure 3-3: Free volume cluster properties for hard sphere (= 0.80) and Lennard-Jones fluids (T* = 1.2, = 0.80). Average values of shape parameters are shown in the legends.

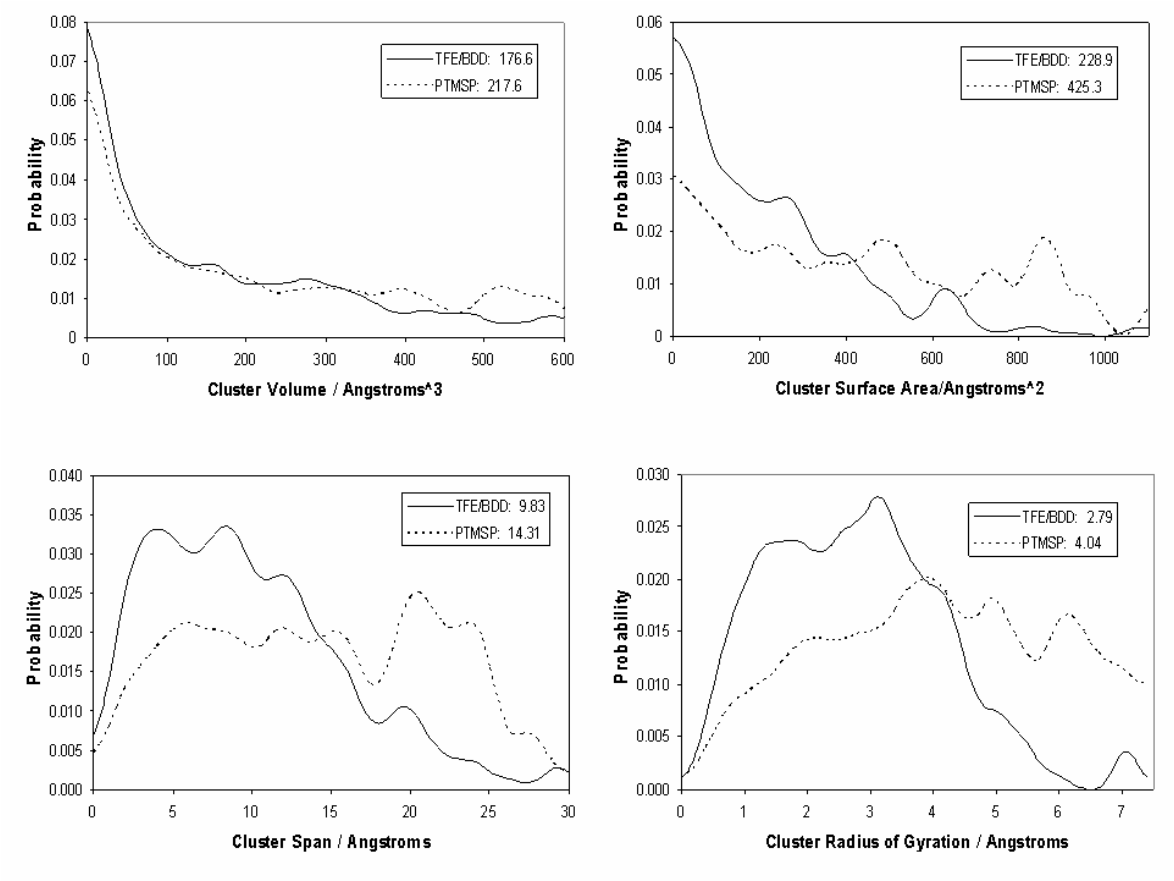


Figure 3-4: Free volume cluster properties for TFE-BDD copolymer and PTMSP. Both are at a density of 0.75 g/cm^3 Average values of shape parameters are shown in the legends.

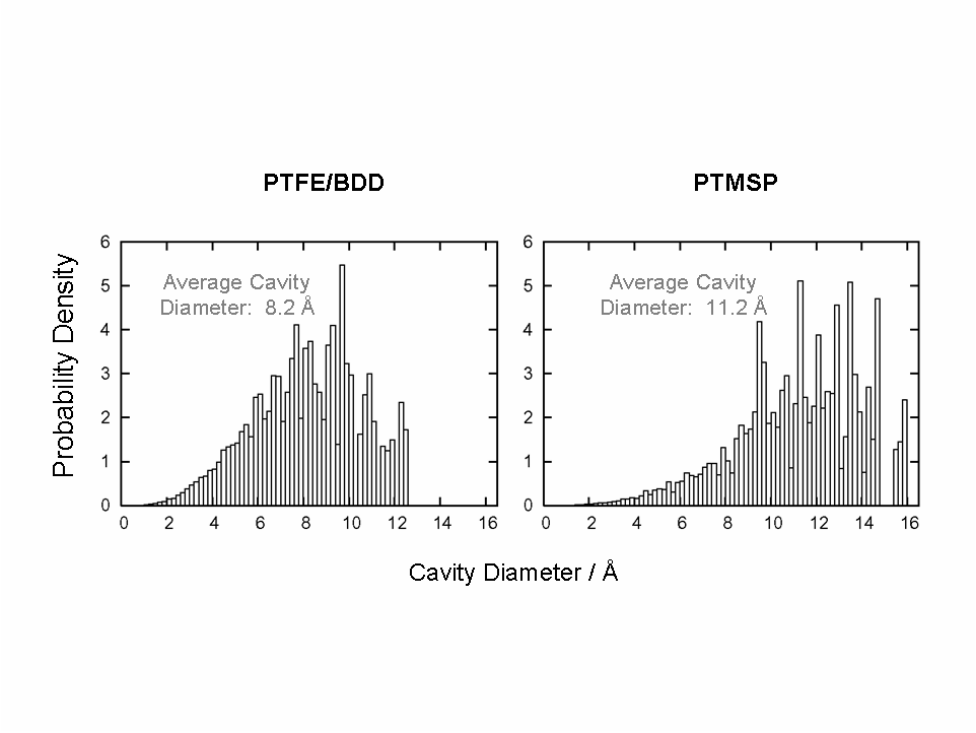


Figure 3-5: Volume-weighted cavity size distributions (without clustering) for TFE-BDD and PTMSP.

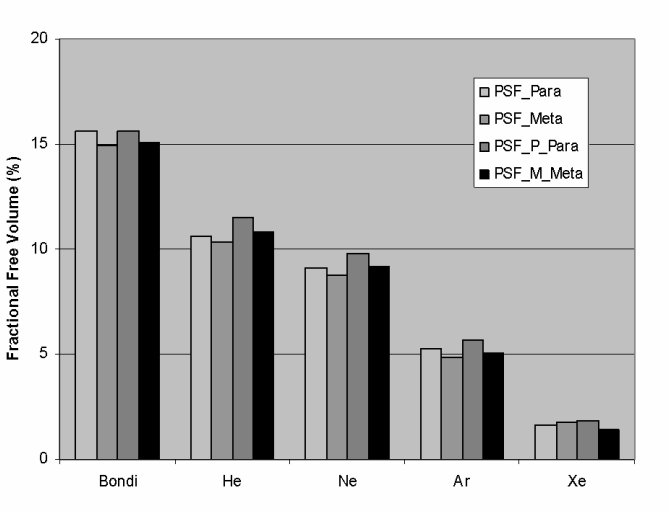


Figure 3-6: Fractional free volume as a function of probe size using ICM compared with Bondi free volume.

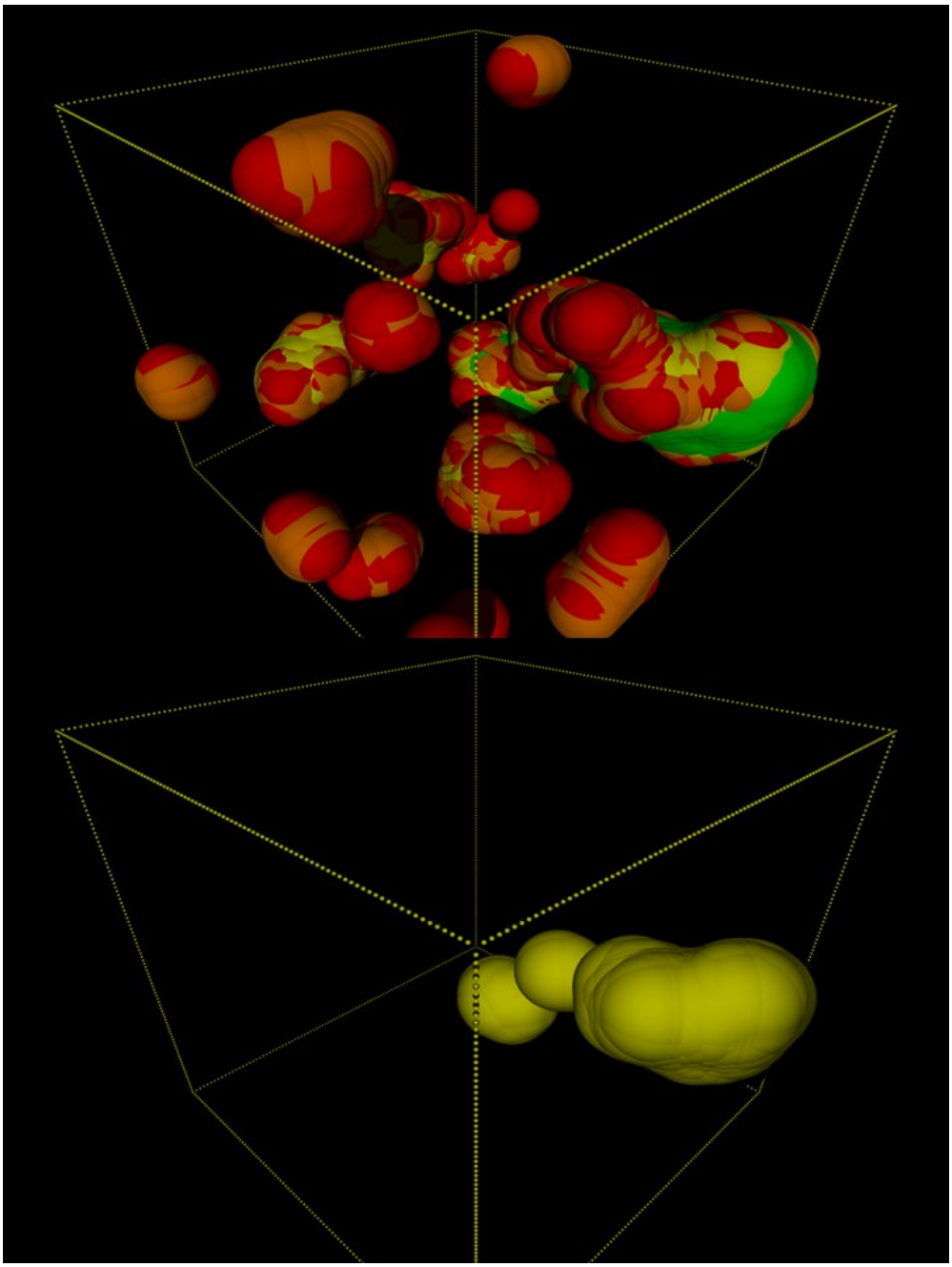


Figure 3-7: Graphical representation of fractional free volume in a sample of TFE-BDD shown for Xenon alone shown below, and for Helium (red), Neon (orange), Argon (yellow) and Xenon (green), shown above.

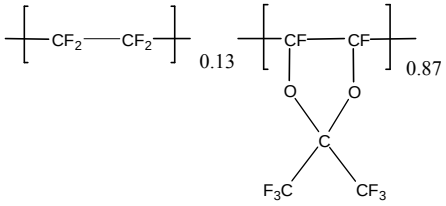
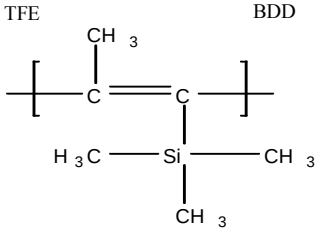
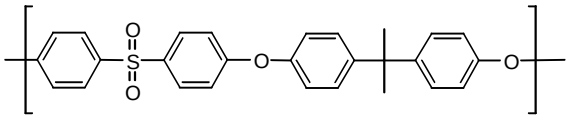
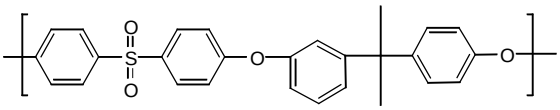
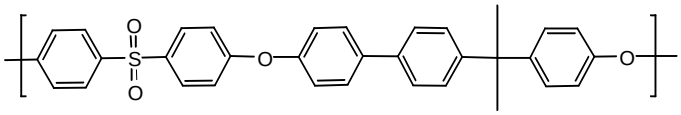
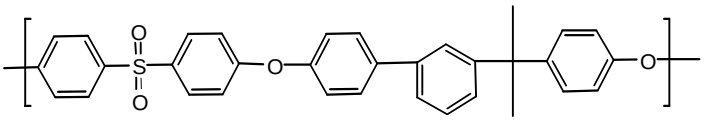
Polymer	Structure	Fractional free volume (Bondi)	Fractional cavity volume (CESA)	Available free volume (He probe)	Available free volume (Xe probe)
TFE/ BDD		0.32	0.132	0.408	0.205
PTMSP		0.29	0.156	0.426	0.352
PSF		0.156	0.076	0.106	0.0162
3,4'-PSF		0.149	0.073	0.103	0.0175
PSF-P		0.156	0.077	0.115	0.0182
PSF-M		0.151	0.076	0.108	0.0147

Table 3-1: Structure and free-volume properties.

CHAPTER 4: POLYMER AGING EFFECTS ON FREE VOLUME

INTRODUCTION

The effects of aging on the free volume distribution in a set of high free volume polymers were studied using the cavity energetic sizing algorithm(CESA)[7, 8] and the cavity overlap technique[2]. Rapid physical aging of Poly(1-trimethylsilyl-1-propyne) (PTMSP), the most permeable polymer known, is characterized by an increase in bulk density and a reduction in observed cavity sizes. Observed experimentally by PALS and theoretically by CESA, this reduction in observed cavity sizes is accompanied by a decrease in the small molecule diffusivity and a reduction in the permeability. The trend is for the larger cavities and also the larger spans of connectivity to disappear with aging. It is unclear whether the actual mechanism of this aging process involves the sudden collapse of the larger holes, the decay of larger holes into smaller holes, or an overall contraction of the material where all holes tend to shrink proportionally as the system ages. Complete results from this work are published in J Phys Chem[3]. Selected results for the cavity overlap properties are included here. All graphs are volume weighted distributions.

MODEL AND SIMULATION DETAILS

BUILDING AMORPHOUS CELLS

The Materials Studio[56] software of Accelrys Inc. was utilized to construct the amorphous packing structure. The COMPASS force field[57] was used in all simulations. The nonbonded interactions of the COMPASS force field include a Lennard-Jones 9-6 function for the van der Waals interaction and a Coulombic electrostatic interaction.

For PTMSP, the initial polymer chain was constructed of 50 repeat units with a 50:50 probability for the occurrence of *cis* and *trans* monomers, mimicking what is believed to be the structure of PTMSP material polymerized in the presence of a $TaCl_5$ catalyst.[3, 19, 37] Figure 4-1 presents the chemical structure of PTMSP. Two PTMSP chains (each with 50 repeat units) were folded in the Amorphous Cell at a density of 0.75 g/cm^3, which corresponds to the well accepted as-cast PTMSP experimental density and at densities of 0.85 g/cm^3 and 0.95 g/cm^3, which represent two aged PTMSP densities. Table 4-1 lists the as-cast and aged experimental PTMSP densities from several references for comparison.[9, 12, 36] The resulting cell lengths for three simulation densities are presented in Table 4-2. Sixty initial states for each density were constructed and followed by 5000 steps of energy minimization to eliminate "hot spots". Afterwards, a 10 *ps* NVT MD run at 298 K was performed for each of the sixty states to equilibrate structures. The resulting equilibrated structures are presumed to be representative of the glassy polymers.

SIMULATION OF CAVITY SIZE DISTRIBUTION

The CESA was then applied to each of the above equilibrated structures. Below is a quick review of CESA:[45-47]

i A polymer structure is created by MD (or MC) simulation.

ii The force field used to generate the structure is replaced with a purely repulsive force field. All atoms remain in fixed positions.

iii A trial repulsive particle is then randomly inserted into the repulsive structure, and a local energy minimum is located in the repulsive force field.

iv After the minimum is determined, attractive interactions are turned on, and the size of the test particle is adjusted until its potential interaction with all other atoms becomes zero. This size is taken as the diameter of a spherical cavity.

v A check is then made to determine whether the initial random inserting point is inside the cavity or not. The cavity is only accepted if the initial point is inside the cavity. This procedure leads to volume distribution rather than a number distribution of cavities.

vi Steps iii to v are repeated enough times to get a representative distribution of cavity sizes for a given structure.

SIMULATION OF CONNECTIVITY AND SHAPE OF THE NANOPORES[58]

By considering the overlap of spherical cavities obtained using the CESA, we also characterize the connectivity of the void spaces. Two cavities are considered to be overlapping if the distance between their centers is less than the sum of their radii. Overlapping cavities are grouped into a new entity called a nanopore (or a cluster). Each cavity belongs to only one nanopore, and each nanopore contains at least one cavity. The connectivity of these nanopores is then characterized in terms of span and radius of gyration, and the shapes of these nanopores are characterized by volume and surface area.

Span: The span is defined as the farthest distance between any two points that lie within the nanopore. Analytically, this is calculated by finding the two cavity centers in the nanopore with the greatest distance between them. The span is this distance between centers, plus the radius of each of the farthest cavity centers.

Radius of Gyration: In general, the radius of gyration of an object is defined as the mass averaged root mean square distance from the center of mass.[58, 59] The radius of gyration of the nanopores in this work was calculated by selecting points at random within the nanopore and

assigning to each of them an equal 'weight'. The center of mass of this set of points was then determined, and the radius of gyration determined as the root mean square distance of the set points in the set from the center of mass of the set. This calculation was performed on a 'per nanopore' basis.

Volume: Volume of the nanopore is calculated by performing a straightforward Monte Carlo integration (Figure 4-3).[52, 58] Points are selected at random from within the simulation box, and are determined to lie within at least one of the component cavities of the nanopre or not. This assures that the volume of overlapping regions is not over-counted. The nanopore volume obtained is the ratio of points inside the nanopore to the total number of points selected, times the box volume.

Surface Area: Nanopore surface area is determined by selecting points that lie on the surface of each component cavity within the nanopore.[58] Sets of zenith and azimuth (longitude and latitude) are selected randomly to generate points uniformly on the surface of each sphere in the nanopore. Zenith angles are sampled from a sinusoidal distribution, in order to prevent divergence in the sampling density at the poles of the sphere. The exposed surface area fraction, ϕ_i, of each component sphere is calculated by counting the number of points which lie on the surface of each sphere (not inside the radius of any other sphere) and dividing by the total number of points sampled on the sphere. The surface area of the nanopore is then determined by multiplying this fraction by the surface area of that sphere σ_i and summing over all spheres. Thus, the nanopore surface area is: $\sigma_{cluster} = \Sigma_i \phi_i \sigma_i$

CONNECTIVITY — SPAN AND RADIUS OF GYRATION

Distributions of spans and radius of gyration values for the nanopores in as-cast and aged PTMSP are presented in Figures 4-2 and 4-3. Both the span and radius of gyration distributions

shift to smaller sizes upon physical aging. The average nanopore span size and radius of gyration, which are tabulated in Table 4-1, also decrease with physical aging. These results suggest that cavities in aged PTMSP are less connected, which may contribute to the decreased permeability and diffusivity in aged PTMSP.

SHAPE — NANOPORE VOLUME AND SURFACE AREA

Distributions of nanpore volumes and surface areas in as-cast and aged PTMSP are presented in Figures 4-3 and 4-4. Both nanopore volume and surface area distributions show an exponential-like decay with increasing nanopore volume and surface area. This is expected since there are always more smaller nanopores than larger ones. There are more smaller nanopores in aged PTMSP than those in as-cast PTMSP. However, there are more larger nanopores in as-cast PTMSP than those in aged PTMSP. PALS results also show a reduction in the number of large free volume elements in aged PTMSP.[63] The average nanopore volume and surface area, which are tabulated in Table 4-1, decrease with physical aging.

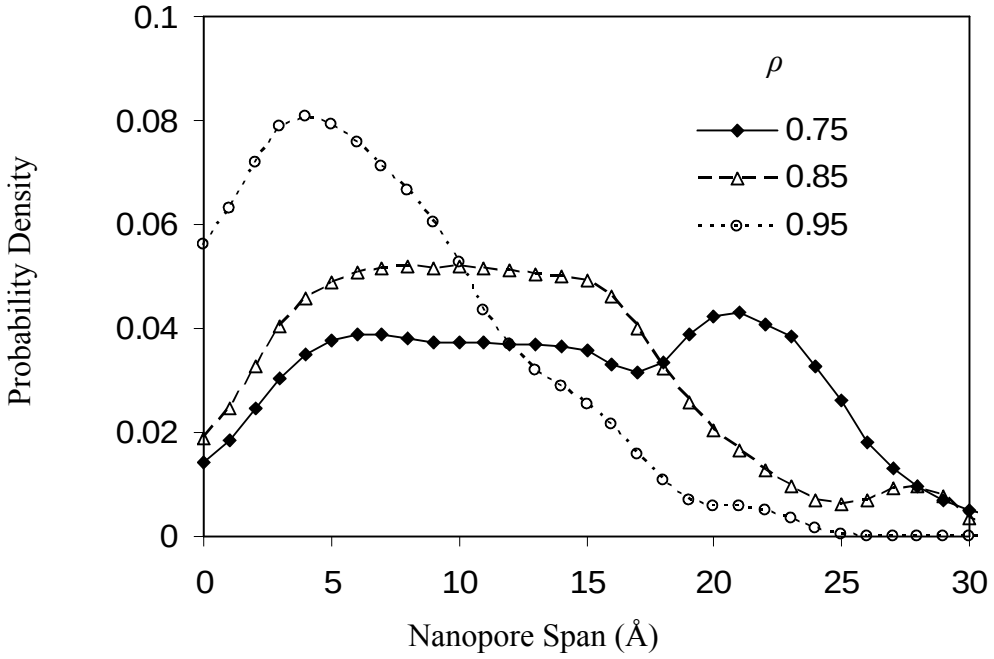


Figure 4-1. Comparison of nanopore span distributions in as-cast (ρ = 0.75g/cm^3) and aged (ρ = 0.85 g/cm^3 and ρ = 0.95 g/cm^3) PTMSP.

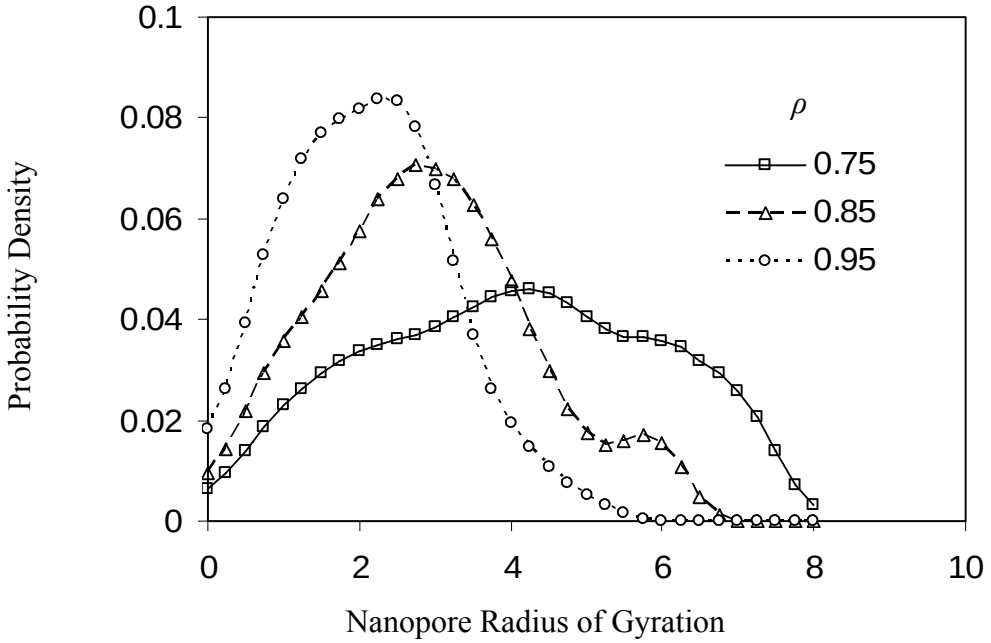


Figure 4-2. Comparison of nanopore radius of gyration distributions in as-cast (ρ = 0.75g/cm^3) and aged (ρ = 0.85 g/cm^3 and ρ = 0.95 g/cm^3) PTMSP.

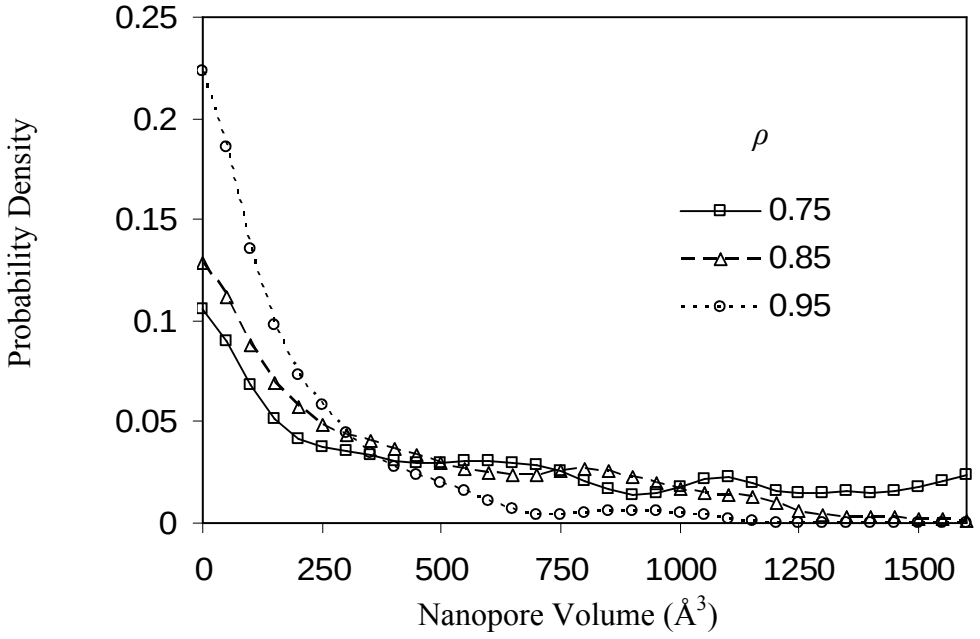


Figure 4-3. Comparison of nanopore volume distributions in as-cast (ρ = 0.75g/cm^3) and aged (ρ = 0.85 g/cm^3 and ρ = 0.95 g/cm^3) PTMSP.

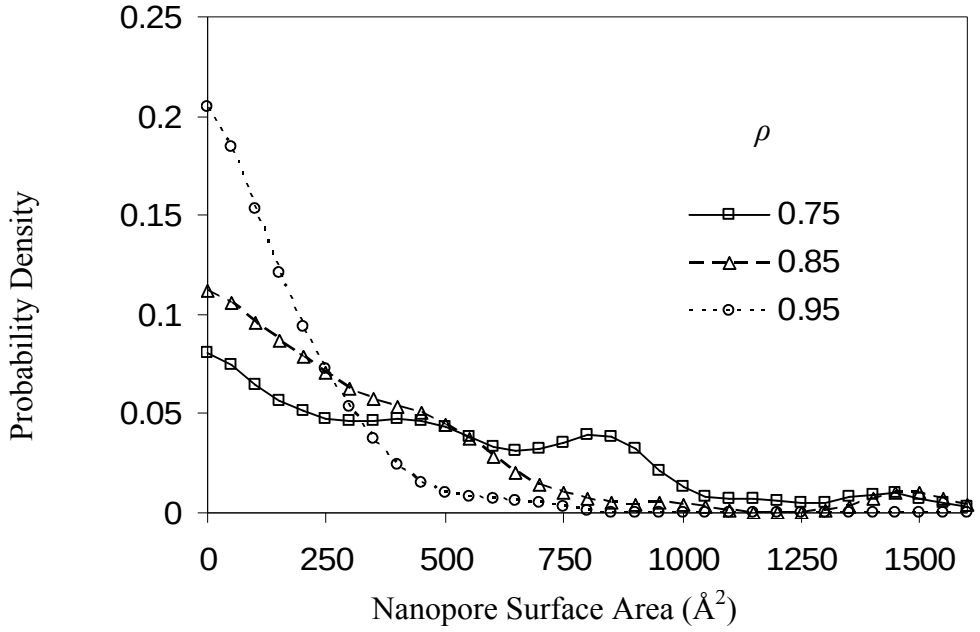


Figure 4-4. Comparison of nanopore surface area distributions in as-cast (ρ = 0.75g/cm^3) and aged (ρ = 0.85 g/cm^3 and ρ = 0.95 g/cm^3) PTMSP.

Polymer	*Average nanopore span size (Å)*	*Average nanopore radius of gyration (Å)*	*Average nanopore volume (Å^3)*	*Average nanopore surface area (Å^2)*
PTMSP (as-cast)	15.06	4.03	557	475
PTMSP-085 (aged)	11.97	2.91	391	325
PTMSP-095 (aged)	7.25	2.12	181	150

Table 4-1: Comparison of Average Span Size, Average Radius of Gyration Size, Average Nanopore Volume and Nanopore Surface Area in As-Cast and Aged PTMSP

CHAPTER 5: PROBABILISTIC MOLECULAR DYNAMICS: SELF-DIFFUSION IN A LENNARD-JONES FLUID

ABSTRACT

A statistical model attempts to simplify the understanding of the self-diffusion process in a 6-12 Lennard-Jones fluid. Particle velocities are sampled from a distribution rather than tracked by direct integration of Newton's equations of motion. Only one particle at a time is allowed to move. Particles change direction suddenly upon collision and in an uncorrelated direction. Some qualitative agreement is obtained between the model and well-documented molecular dynamics results.

INTRODUCTION

Self-diffusion is the motion of the component molecules of a pure fluid traveling within the pure fluid. It cannot be measured experimentally because the molecules are inherently indistinguishable, and any effort to observe them directly would require systematic alteration of the system and its motions, a direct consequence of the Heisenberg principle. Molecular dynamics, however, allows us to observe the motions of individual molecules non-intrusively. The self-diffusivity is determined by watching a large enough population of molecules of the fluid for a long enough time to obtain a statistically consistent value. It is defined in terms of the Einstein relationship[74]:

$$\left\langle R^2 \right\rangle = 6Dt$$

where $<R^2>$ represents the mean-square displacement of all molecules from their starting points and t the time elapsed.

Although representative configurations of the fluid have in the past been satisfactorily generated using Monte Carlo techniques, there is no accepted way to assign time values which describe the transformation of one configuration into another, and to thereby capture the inherently time dependent nature of diffusion. What is undertaken here is a mechanism to relate timeless Monte Carlo simulation data to an implicit time scale. The technique is illustrated by its application to a Lennard-Jones fluid. Meier et. al.[75-77] have generated an exhaustive array of self-diffusion coefficients for the Lennard-Jones 6-12 fluid using molecular dynamics. These results are well-documented and serve as the point of comparison for the results generated by the technique introduced here.

THE TECHNIQUE

The technique is developed around the following observations. Consider a particle in a fluid moving amongst many other identical particles of the fluid:

1. It travels along a *curved* trajectory from one point of collision until another point of collision because it is *continuously* affected by the surrounding forcefield.
2. There are a range of distances traveled between collisions.
3. Its velocity changes with every collision.
4. There is a well-known distribution describing the possible velocities that the particle may display, a characteristic of the temperature of the system.

The following approximations are made:

1. The particle travels along a *linear* trajectory from one point of collision until another point of collision and is *unaffected* by the surrounding forcefield.
2. The distances traveled between collisions are uncorrelated.

3. Its velocities before and after a collision are uncorrelated and may be sampled from the overall distribution of velocities.

4. Its velocity between collisions remains constant.

The technique works by compiling a list of typical path length/velocity pairings for a given temperature and density. Each of these is generated by selecting one particle at random from the fluid configuration, assigning a speed and direction to it, and moving it forward and/or backward (see table 5-1) until it reaches a collision point. The other particles in the fluid remain fixed in place.

Since the distribution of kinetic energies of particles obeys Boltzmann statistics, an appropriate kinetic energy is selected by sampling from the Boltzmann distribution:

$$P(KE) = e^{-KE/kT}$$

By selecting a continuous random variable R, $0 < R < 1$ An appropriately sampled kinetic energy for the particle at the temperature T is:

$$KE = -kT \ln R$$

The speed is taken from this kinetic energy as:

$$v = \sqrt{\frac{2KE}{m}}$$

A direction is selected at random by specifying azimuthal and zenith angles:

$$\phi = 2\pi R$$

$$\theta = \cos^{-1}(2R - 1)$$

The use of the arccosine function insures that the values of theta thus sampled are not biased toward the poles. This combination of speed and direction constitute a velocity. The direction of this velocity is the same as the direction of displacement.

A collision point is defined as the point along the particle's trajectory where the particle's kinetic energy becomes less than the potential energy of interaction with the remaining frozen particles in the system.

The total simulation time elapsed is a sum of the time taken for each of these linear paths:

$$t = \sum_i \vec{l}_i \bullet \vec{v}_i$$

The overall displacement $\vec{R}$ is taken as the vector sum of these paths:

$$\vec{R} = \sum_i \vec{l}_i$$

The self-diffusion coefficient is then obtained by the Einstein[74] relation:

$$6Dt = \left\langle \vec{R} \bullet \vec{R} \right\rangle = \left\langle R^2 \right\rangle$$

MD	This curve represents the molecular dynamics results of Meier, et al[75-77]
2F	The particle is moved forward from its initial point until it reaches its forward collision point. The value of l_i used is twice this distance.
BF	The particle is moved forward and backward along the trajectory until it reaches forward and backward collision points. The distance between the two collision points is used for l_i .
2P	Same as BF, however the kinetic energy required for the collision is increased by sampling another energy from the Boltzmann distribution. This is to compensate for the energy of the particle that is colliding with the test particle.
2P-0_2	Same as 2P, but a random variable R = (-0.1σ, 0.1 σ) is added to l_i to compensate for uncertainty in the collision point.
2P-1_0	Same as 2P-0_2, but R = (-0.5 σ, 0.5 σ).

Table 5-1: Description of the rules used to generate the various curves.

SIMULATION DETAILS

A series of configurations for a 6-12 Lennard-Jones fluid were generated by a Monte Carlo simulation. Reduced temperatures of 1.5, 3.0, and 6.0 and reduced densities from 0.1 through 1.0 in increments of 0.1 were simulated. Each simulation used 1372 particles with periodic boundaries, as was done by Meier[75-77].

To generate an appropriate initial condition, the simulation box was divided in half along the x and y axes, and populated by randomly placing 343 molecules inside. A simulation of 10000 monte carlo steps was performed on this smaller system to generate an appropriate near-equilibrium initial condition. This system was then replicated along the x and y axes to create the system of 1372 particles. The larger system was then allowed to evolve for 5000 steps before any configuration was captured for analysis, and then again for 5000 steps between subsequent captures. A total of 6 configurations were captured for each density point observed. A variable

step size was used for the displacements of the particles in the system, in order to maintain an acceptance ratio of approximately 20% of attempted moves.

Probabilistic molecular dynamics simulations were performed on the configurations thus generated. The simulations were allowed to run for a cumulative simulated time of 10.0 nanoseconds. Figure 5-1 shows the diffusivity value obtained as a running average vs simulation time at a reduced density of 0.1 and reduced temperature of 6.0.

RESULTS AND DISCUSSION

Most notable is that these simulations were able to run for a cumulative time of 10 nanoseconds. Typically, molecular dynamics simulations require on the order of 1000 times as much computer time to achieve a similar result. This folding of time is due jointly to the fact that only one particle is moving at a time in the simulation, as well as to the absence of exhaustive bookkeeping of equation of motion data.

The BF (backward then forward) technique is the most intuitive of the path definitions presented, and consistently captures diffusivities to within 25% of the published molecular dynamics values. The deviation from these published values is, however systematic, underpredicting diffusivities, especially at higher temperatures. 2P is a reasonable enhancement to BF, in that it samples energies of two molecules (most collisions involve two molecules) which effectively extends the mean path length slightly, thus providing a higher diffusivity.

The 2P-0_2 and 2P-1_0 add a small random deviation on the order of the particle diameter to the path length, in order to account for (very crudely) an uncertainty in where the collision occurs. This results in a small increase in the overall mean-square displacement, and thus in the diffusivity, but not in any consistent way. In fact, the random component becomes

much more important as the density increases and intercollision distances decrease. At high density, the path length distribution is almost entirely accounted for by the random variable.

2F, in which the forward path length is simply doubled to estimate the total forward and backward distance, is included because the numbers thus generated are consistently the closest to the molecular dynamics results. Although the forward distances and backward distances share the same distribution, the fact that we are interested in the mean of the square of sums of these distances, which is larger than the mean of the sum of the squares, and thus 2F overpredicts BF.

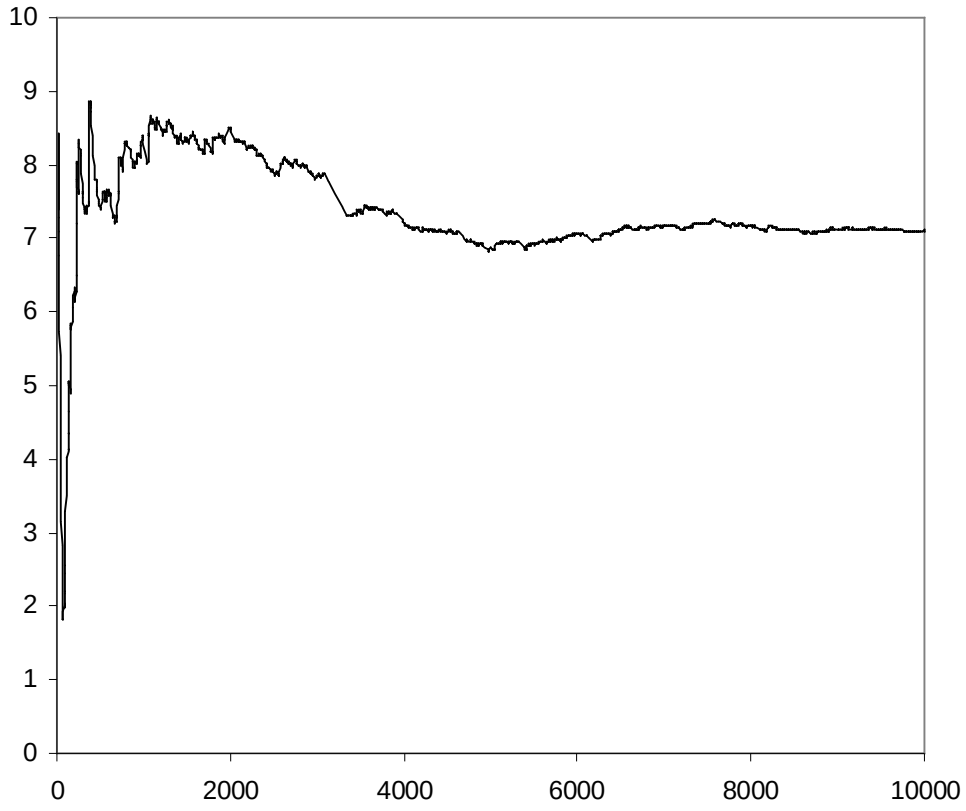


Figure 5-1: Running average of diffusivity obtained for density 0.1, T*=6.0 vs time in picoseconds.

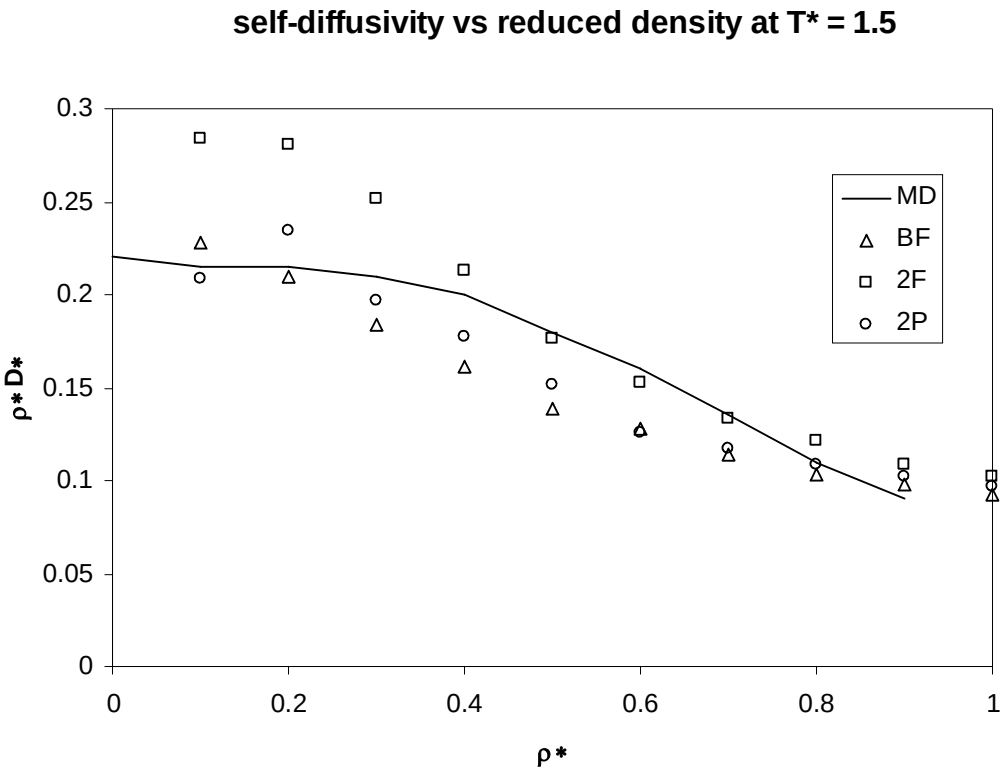


Figure 5-2: The BF and 2P curves give the best results at this temperature, with the 2P being closest to the published molecular dynamics values.

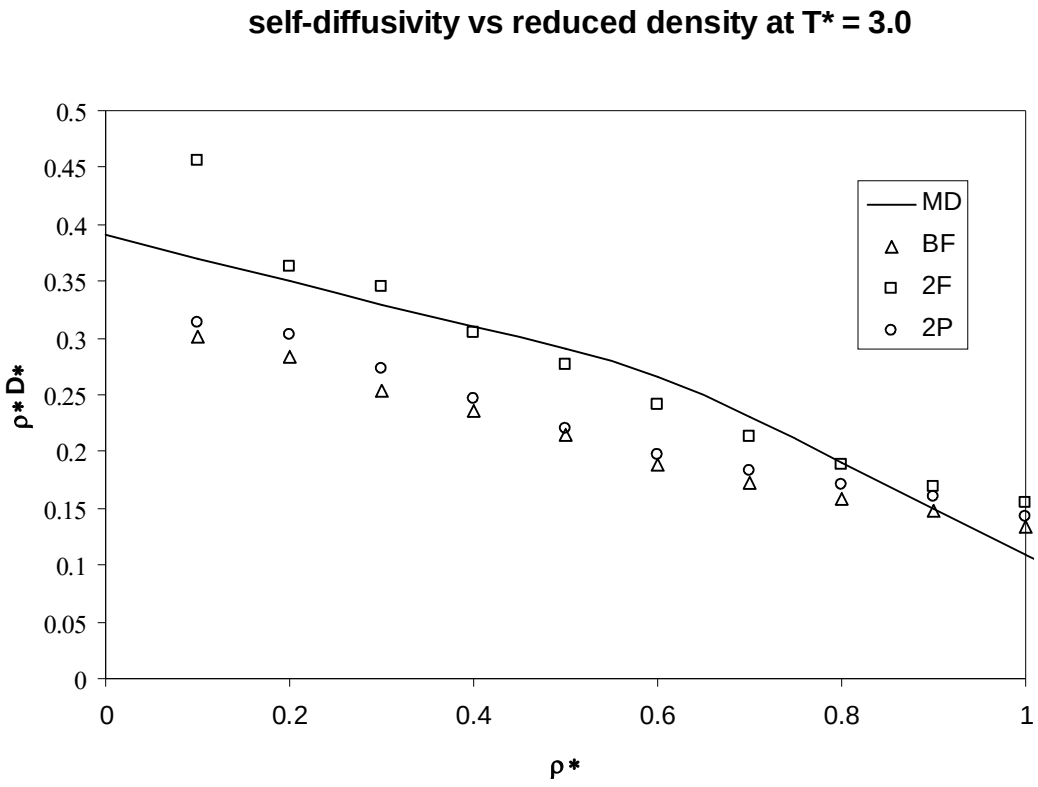


Figure 5-3: As temperature increases, the monte carlo results increasingly underpredict the molecular dynamics results.

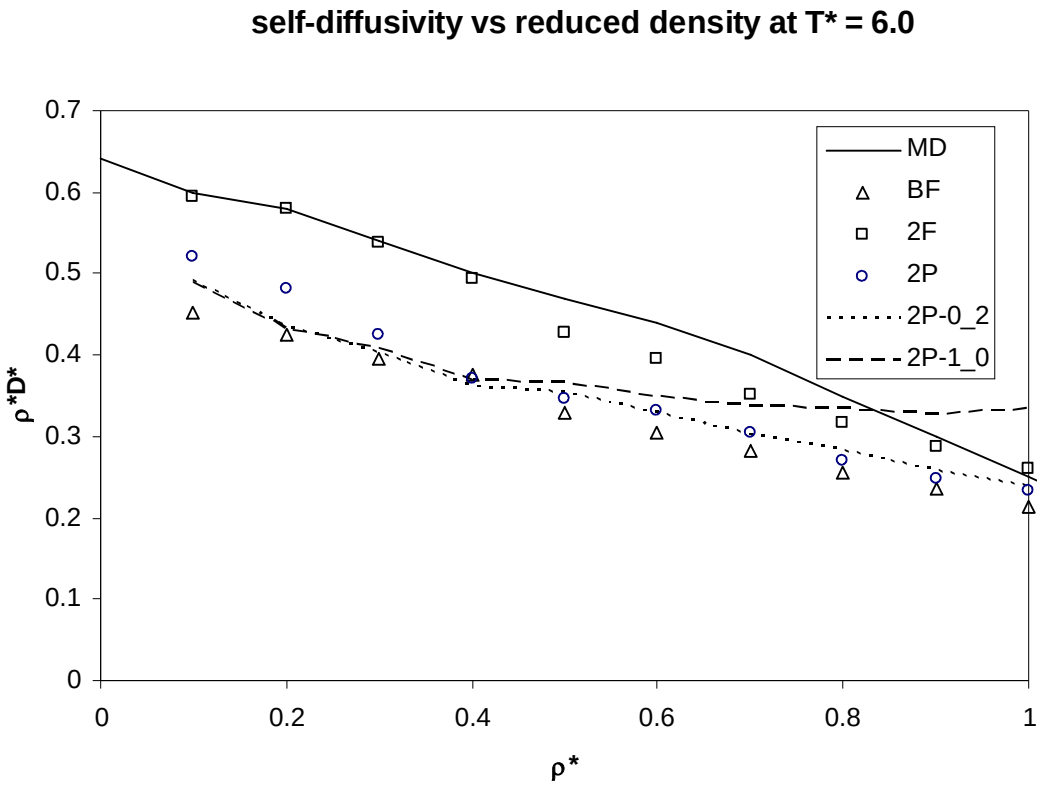


Figure 5-4: At the highest reduced temperature studied, the monte carlo results continue the trend of increasingly underpredicting the self-diffusivity. The enhancement of adding the random uncertainty term in the 2P-0_2 and 2P-1_0 series offsets this only slightly at low densities where the result is most off, and actually overshoots the molecular dynamics results at higher densities.

CHAPTER 6: SMALL MOLECULE DIFFUSION BY PROBABILISTIC MOLECULAR DYNAMICS: MACROSCOPIC DIFFUSIVITY FROM MICROSCOPIC DATA USING A MESOSCALE MODEL

ABSTRACT

A set of techniques which makes possible the calculation of the diffusivity of light gases in an amorphous material from limited Monte Carlo and molecular dynamics data is presented. A 'conduction' average, distinct from linear and reciprocal averaging and appropriate to diffusion problems, is defined. A means of obtaining diffusion coefficients from Monte Carlo data is presented for the first time. The techniques are demonstrated by calculating diffusion coefficients from molecular dynamics and Monte Carlo data of helium and methane in polystyrene, and of helium and neon in three pairs of polysulfone isomers. Results include diffusion coefficients as small as 1.0×10^{-9} cm^2/s, too small to be calculated using traditional molecular dynamics. The results are in close agreement with traditional molecular dynamics results where available and also with experimental data.

INTRODUCTION

Molecular dynamics as it exists stands as a tribute to the comprehensive understanding of all that is known about the motions and behaviors of atoms and molecules from an ab-initio standpoint. It is at once elegant and physical. It is detailed and thorough. What it may not always be is efficient. Introduced here is a set of probabilistic enhancements to the technique, hereafter called Probabilistic Molecular Dynamics (PMD), which lumps together a set of excruciatingly detailed observations into simpler statistical descriptions of expected behaviors.

Diffusant velocities are replaced with a random variable sampled from an appropriate velocity distribution. Collision details are replaced with a simple energetic criterion. And the carefully calculated trajectories between collisions with the polymer matrix are replaced with a random choice of direction. Motion of the polymer matrix is accounted for by coarse-graining the long-time behavior in the multiscale model. These assumptions eliminate computationally expensive bookkeeping and allow appropriate approximations to extend molecular dynamics to time scales heretofore inaccessible because of the amount of computation time required. The technique is illustrated by measuring a set of diffusion coefficients several orders of magnitude smaller than what has previously been measured, and with good agreement to experimentally measured values. A range of diffusivities from 10^{-5} to 10^{-9} cm^2/s was observed.

Sok et al[42] have effectively demonstrated the strength of molecular dynamics for predicting the diffusivity of small molecules in a polymer, with results that strongly reflect the importance of the motion of the polymer matrix, as it opens, closes, and reshapes voids which allow penetrant to travel. Tsige and Crest[43, 44] have used bead-spring models of polymers to look at the motion of solvent using molecular dynamics, and have observed Fickian diffusion over a range of diffusivities. Their model also accounts for motion of the matrix.

Zielinski and Duda[51] developed a model to predict small molecule diffusivity without experimental data, based on the joint probability of finding a sufficiently large void into which to move and the possibility of having the energy to escape from the current position. Their model produced surprisingly good results considering that it does not take into account the specifics of hole morphology and connectivity for particular polymers. Later studies consider the role of thermal fluctuations[52] and local geometry[53, 54]. Han and Boyd[55] look at methane in polystyrene

as a series of hops from pore to pore, using jump lengths and probabilities to characterize the diffusion mechanism.

Coarse-graining and the approximation of distributions with appropriately sampled values as inputs to a stochastic model has found wide application. The 'equation-free' approach of Rico-Martinez et al.[45] uses a coarse-grain projective stochastic model to reproduce the limit cycle behavior of a monomer-dimer-poison surface model[46] of Vigil et al. Gauthier and Slater have examined diffusion through a heterogeneous lattice by implementing blocking sites and external fields[47-49]. Armatas, et al[50] have used a Monte Carlo scheme in populating a network with pores of different types to obtain an effective pore diffusivity for the composite material[50], yielding normal Fickian diffusion. Of recent interest has been the development of a fractional equation approach to normal and anomalous diffusion[78].

PART I: CONDUCTION AVERAGING

Different means of calculating an average apply in different physical situations. Reduced mass and reduced charge are a familiar type of average used in astrophysics and electrodynamics, respectively. A reciprocal average is used in circuit theory to calculate the equivalent resistance of a set of resistors in parallel:

$$\frac{1}{R_{eq}} = \frac{1}{R_1} + \frac{1}{R_2} + \ldots$$

In fact, this same reciprocal average can be applied analogously to any of the well-known transport phenomena. A list of analogous quantities between mass and charge transport is shown in table 6-1. Electrical conduction also reflects the transient phenomena associated with the inductance, capacitance, etc. which are relevant in circuit theory but are not important when considering only the steady-state properties.

Mass Diffusion	Electrical Conduction
Mass Flux: $\vec{N}''$	Current Density: $\vec{J}$
Diffusivity: D	Conductivity: σ
Mass transfer resistivity: $\rho_{mass} = \frac{1}{D}$	Resistivity: $\rho = \frac{1}{D}$
Concentration: C	Electrical Potential: ϕ
Fick's law: $\vec{N}'' = -D\nabla C$	Ohm's law: $\vec{J} = -\sigma\nabla\phi$
Mass transfer resistance: $R_{mass} = \frac{l}{DA}$	Electrical Resistance: $R = \frac{l}{\sigma A}$

Table 6-1: Mass and charge transfer analogous quantities and phenomenological ldaws.

Conduction averaging involves taking a number of sample configurations of a polymeric material, each with its own diffusion coefficient relative to a particular diffusant, and forming a model for the behavior of the macroscopic material by treating it as a composite of these separate configurations. By randomly arranging cubic blocks of the sample configurations on a lattice (figure 6-1) and determining diffusion rates through the composite material, a self-diffusivity for the composite is produced. There is obviously a drastic increase in the complexity of the solution compared to a 1-D model, and it is intuitive that if the conductivities are not all equal in the composite, that the penetrant will find the path of least resistance when traveling through our solid. A closed-form solution to this problem may indeed exist, predicated on further assumptions about the size of the solid and where we start and stop the penetrant's journey, however a brute-force Monte Carlo scheme has been implemented here with good results.

To obtain input values for the conduction average, a series of simulations are performed, either by traditional molecular dynamics or by the PMD scheme described below, of a light gas

diffusing in a porous polymer matrix. Performed at the molecular or ab-initio level, these return a diffusivity value based on molecular scale motion. In materials with notable opening and closing of channels, the values obtained can vary by several orders of magnitude. A linear average over sample configurations can skew the average diffusivity value accordingly. The conduction average takes the heterogeneous nature of the medium into account both elegantly and intuitively when calculating the diffusivity.

The mesoscale simulation is performed by treating the motion of a diffusant through the composite system as a process of hopping from site to site, using the formalism of a continuous time Markov process. Since we are interested in the infinite dilution case, we consider the system with only one diffusant particle in it at a time, and the state of the system is equivalent to the location of the diffusant. We consider each site to be connected to six other adjacent sites in the cardinal directions, with hops occurring only between adjacent sites. A hopping probability, proportional to the diffusivity of the component structures, is used to determine the likelihood that a molecule will jump from one cube of material to another along the lattice. By determining how many steps are required to achieve the same mean square displacement through this composite matrix, an effective diffusivity is obtained.

This selection of hopping probabilities can be attributed to a theorem from probability which states that the one-step transition probability from a given state A to another state B is equal, in the limit of a large number of trials, to the *fraction* of trials which will go from A to B. Similarly, the *fraction* of molecules which will go from a given container A to another container B over a measured time step varies inversely proportional as the mass transfer resistance between the containers, or equivalently, the fraction that hops to the other container is proportional to the diffusivity. Thus the diffusivity as a proportion of molecules crossing the boundary between

containers is translated to a single-hop probability between adjacent states. When translating the diffusivities into probabilities, they are scaled by the maximum diffusivity; this is done in order to enhance the efficiency of the algorithm and does not affect results.

We are interested in the time required to obtain some mean-square displacement. We define our effective diffusivity D_{eff} in terms of this mean-square displacement $\langle R^2 \rangle$ and the time t_{eff} required to achieve it using the Einstein relationship:

$$\langle R^2 \rangle = 6 D_{eff} t_{eff}$$

We condition our outcome by comparing the time required to obtain the same mean-square displacement when all diffusivities are equal, specifically when they are equal to the maximum diffusivity value $D_{\max}$:

$$\langle R^2 \rangle = 6 D_{\max} t_{\max}$$

When all diffusivities are equal, this reduces to the problem of random walk on a cubic lattice. Since we are considering the *same* mean square displacement $\langle R^2 \rangle$, we can equate the two, cancel the factor of 6 on each side, and write an expression for D_{eff} in terms of $D_{\max}$ and the amounts of time to achieve each result :

$$D_{eff} t_{eff} = D_{\max} t_{\max}$$

Let Δt represent the time step between hopping attempts. If N_{eff} represents the number of attempts required to obtain the mean square displacement $\langle R^2 \rangle$ when $D = D_{eff}$, and N represents the number of attempts required when $D = D_{\max}$, then the times required to obtain

the same probability density profile are $t_{eff} = N'\Delta t$ and $t_{\max} = N\Delta t$. Inserting into the last equation we have:

$$D_{eff} N_{eff} \Delta t = D_{\max} N \Delta t$$

Solving for D_{eff}, we have:

$$D_{eff} = D_{\max}\left(\frac{N}{N_{eff}}\right) \qquad (1)$$

What this reveals is that we can develop an expression for the effective diffusivity based on the acceptance ratio for hopping moves. When all diffusivities are equal, and thus equal to Dmax, then N' is equal to N, and the effective diffusivity is the same value which was input.

Two levels of model detail are presented for generating hopping probabilities: In the first, the hopping probability is directly proportional to the diffusivity of the destination cell. In the second, the hopping probability is an appropriate combination of the diffusivities of the source and destination cells. While the second may seem more physical, it can be shown to be equivalent to the first, except that there are $O(N^2)$ inputs instead of $O(N)$, due to the cross terms. We were able to obtain good results using only the simpler one-site model.

SITE APPROXIMATION

Consider the states of a particle hopping about from cube to cube in our matrix of samples. It can either:

- Stay where it is
- Jump to a cube of material of a different type
- Jump to a cube of material of the same type

Setting this process up as a Markov chain, we denote the configurations with the letters A, B, C, etc.. and designate the case where the particle jumps from one cube to another with an asterisk. Thus the transition AA means the particle is in a cube of A and stays where it is, AA' means the particle has jumped to an adjacent cube of A, AB' means that the particle has jumped to an adjacent a cube of B, and AB indicates the particle staying put while simultaneously jumping to a cube of type B, which is of course impossible and is represented by a 0 probability in the transition matrix:

$$P = \begin{array}{c} \\ A \\ B \\ \vdots \\ A' \\ B' \\ \vdots \end{array} \begin{array}{c} \begin{array}{cccccc} A & B & \cdots & A' & B' & \cdots \end{array} \\ \begin{bmatrix} P_A & 0 & \cdots & D_A & D_B & \cdots \\ 0 & P_B & \cdots & D_A & D_B & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \cdots \\ P_A & 0 & \cdots & D_A & D_B & \cdots \\ 0 & P_B & \cdots & D_A & D_B & \cdots \\ \vdots & \vdots & \cdots & \vdots & \vdots & \cdots \end{bmatrix} \end{array} \frac{1}{\eta D_{\max}}$$

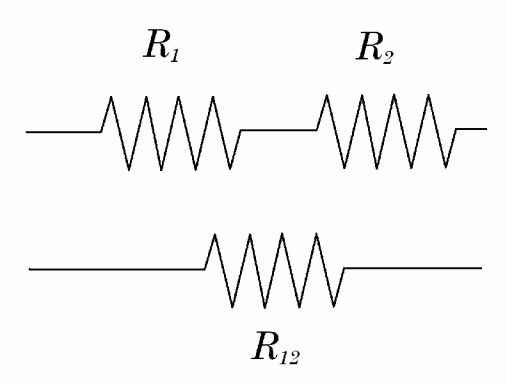
Where η is the number of samples and $D_{\max}$ is the largest diffusivity value. $D_{\max}$ is used so that the probabilities will add to 1, as is required for the transition matrix. The P_A, P_B, etc. represent the probability of the particle *not* moving in a given turn and will be represented by the unsubscripted symbol P. Notice that because each row must add to unity, all of these will be equivalent.

PAIR APPROXIMATION

Diffusion, or mass transport, obeys the same class of phenomenological laws observed for the other transport phenomena: heat transfer, momentum transfer, and electrical conduction. The analogy to electrical conduction lends itself readily here if we employ the idea of a diffusion

'circuit' to our problem. Note that diffusivity, conductivity, and resistivity represent intensive material properties whereas resistance is an extensive property, characterized by size and geometry of the material.

To approximate a net diffusivity between two nodes, we describe two mass conductors of length l and cross sectional area A placed in series and draw a mass conduction circuit:



Here, $2l$ represents the cube dimension. The resistances for each part are:

$$R_{mass,1} = \frac{l}{D_1 A} \text{ and } R_{mass,2} = \frac{l}{D_2 A}$$

Since resistances in series add, the total mass resistance is:

$$R_{12} = R_1 + R_2$$

or:

$$\frac{2l}{D_{12} A} = \frac{l}{D_1 A} + \frac{l}{D_2 A}$$

Solving for D_{12}:

$$D_{12} = \frac{2 D_1 D_2}{D_1 + D_2}$$

These pair values are used to populate the transition matrix:

$$P = \begin{array}{c} \\ A \\ B \\ \vdots \\ A' \\ B' \\ \vdots \end{array} \begin{bmatrix} \\ P_A & 0 & \cdots & D_{AA} & D_{AB} & \cdots \\ 0 & P_B & \cdots & D_{BA} & D_{BB} & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \cdots \\ P_A & 0 & \cdots & D_{AA} & D_{AB} & \cdots \\ 0 & P_B & \cdots & D_{BA} & D_{BB} & \cdots \\ \vdots & \vdots & \cdots & \vdots & \vdots & \cdots \end{bmatrix} \frac{1}{\eta D_{\max}}$$

In the pair approximation, the P_i values in the transition matrix are not all equivalent. Since each row of the matrix must sum to one, we have:

$$P_i = 1 - \sum_{j=1}^{\eta} \frac{D_{ij}}{D_{\max}}$$

As before, η represents the number of sample configurations, and P_i represents the probability of *not* moving in a given time step, if the diffusant is currently in a sample of type i. $D_{\max}$ now represents the largest *pair* diffusivity value.

MEAN-FIELD TREATMENT

Equation 1 tells us that we can determine our composite diffusivity value from knowledge of the maximum component value and the ratio of accepted moves. In the site approximation, the probability of moving in a given turn is equal to $1-P$, P being the probability of staying put:

$$1 - P = \frac{1}{\eta D_{\max}} \sum_{i=1}^{N} D_i$$

The probability of staying put is equal to:

$$P = \frac{N' - N}{N'} = 1 - \frac{N}{N'} \text{ or } \frac{N}{N'} = 1 - P$$

which, using equation 3 gives:

$$D_{eff} = D_{\max}\left(\frac{N}{N'}\right) = D_{\max}\left(\frac{1}{\eta D_{\max}}\sum_{i=1}^{N} D_i\right) = \frac{1}{\eta}\sum_{i=1}^{\eta} D_i$$

Thus the site approximation simply gives a linear average of the D's which are input, because the P's are all the same. For the pair approximation, mean field treatment returns a linear average of the reciprocal averages of all possible pairings of cells. The inadequacy of a mean field model can be shown most vividly when considering the 1-dimensional case: Consider a large array of more or less conducting elements arranged on a line in no particular order. Now consider the addition of a single insulating element. What is the net conductivity of the system? In considering a linear average of the conductivities, there is a finite value, approximately the same as the value when the insulator is absent. But clearly no transport occurs due to the presence of the single insulator. In the case where there are two dimensions, the overall conductivity does not drop to zero, as it is possible to circumnavigate the insulating element, however, this takes several additional hops. In 3 dimensions, there are still more potential pathways around the insulator. This illustrates the inherent inadequacy of either mean field model to capture even this simple nuance of diffusion.

LIMITING PROBABILITY:

Consider the limiting behavior of this system in terms of the limiting probability of the transition matrix: By definition of the transition probabilities, a diffusant is more likely to jump into a cell of higher diffusivity from a cell of lower diffusivity than the reverse. This means that after a long time (or many transitions) there would be a greater probability density of finding the diffusant in the cell with higher diffusivity, or at a higher 'concentration' if we allow more than one molecule at a time to diffuse. It is interesting that a region of greater free volume displays

such a direct correlation with greater solubility (and higher concentration) as well as greater diffusivity.

SIMULATION DETAILS

A cubic lattice of dimensions 2^8 x 2^8 x 2^8 is populated by selecting at random from the set of diffusivity values supplied. No conservation principle is applied, i.e. for each lattice point, a value is chosen at random from all of the values supplied. A periodic boundary condition is implemented by using a 32-bit integer type for the x, y, and z coordinates, but considering by way of a logical AND only the least significant 8 bits of the dimension, efficiently extending the number of possible locations to 2^{32} x 2^{32} x 2^{32} using only 2^8 x 2^8 x 2^8 points. Such treatment is used to more effectively consider phenomenological/macroscopic behavior through proper scaling.

The following procedure is repeated 10 000 times:

1. A site is selected at random and without bias from all lattice sites as the insertion point for the test particle.
2. 1000 attempts are made to move the particle:
    a. A direction is selected at random from the 6 cardinal directions.
    b. A random number is selected such that it is at least 0 and at most 1.
    c. If the ratio of the diffusivity of the destination lattice point to the maximum diffusivity is less than the random number selected in b, then the move is accepted.
3. Once the particle finishes its journey, the square of its displacement from its introduction point is calculated.

By calculating the mean-square displacement after all trials have completed, a diffusivity value is returned by comparing this mean-square displacement achieved to the mean-square displacement achived when all diffusivities are equal to $D_{\max}$:

$$\frac{6D_{eff}t}{6D_{\max}t} = \frac{\langle R^2 \rangle}{\langle R_{\max}{}^2 \rangle} = \frac{\langle R^2 \rangle}{N\lambda^2}$$

Considering a unit lattice has $\lambda = 1$ and eliminating the factors of 6t, we have:

$$D_{eff} = \frac{\langle R^2 \rangle}{N} D_{\max}$$

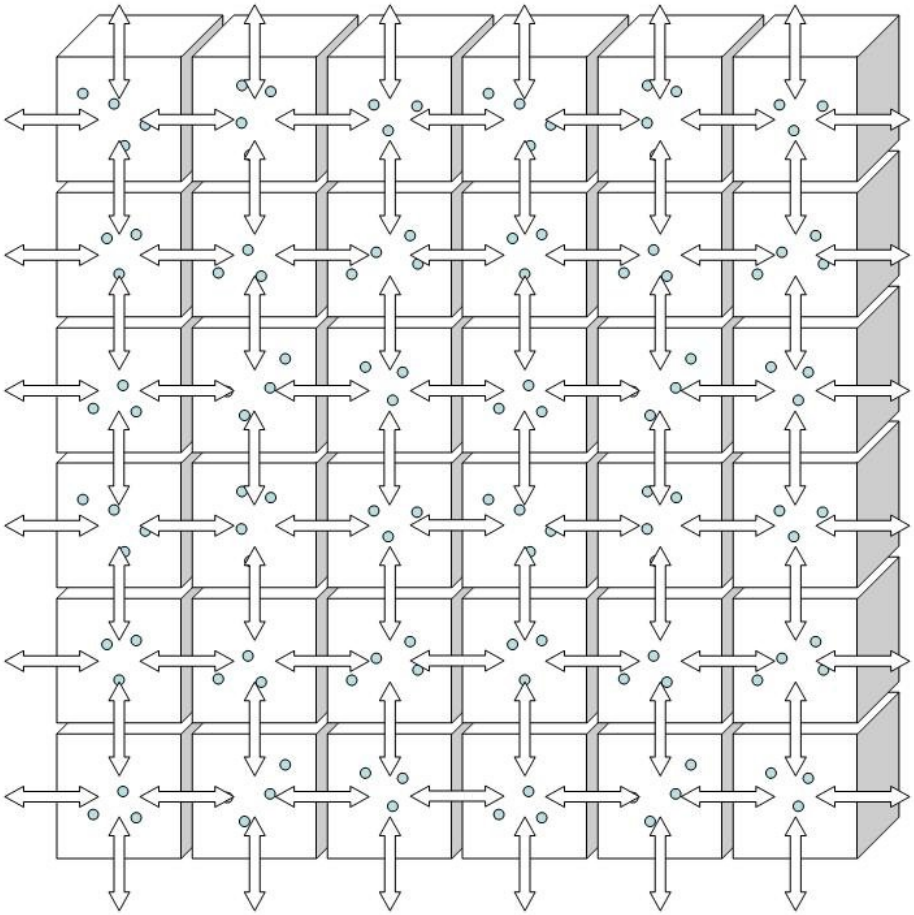


Figure 6-1. Conduction averaging uses an infinite grid of containers and hopping probabilities to determine an effective conductivity for a composite material as if it were made of these individual cells.

TEST CASES

A set of test cases is presented here to illustrate the method. Figure 6-2 shows that conduction averaging represents a distinct solution from linear and reciprocal averaging. The conduction average is higher over the entire range of diffusivity combinations presented, because the technique allows for 'channelling' along paths of lesser resistance, a typical transport behavior.

Providing a large set of values [1, 1, 1, …, 1, 1] will return a value of 1, whereas the set of values [1, 1, 1, …, 1, 1, 2] will return a diffusivity of approximately 3/2 (see figure 6-3). This is not intuitive, as one would not expect that one outlier value would not offset the entire set of values in such a way, but this is fact exactly what happens. Even in the limit of a very large set of values, this non-asymptotic behavior persists as a consequence of the limiting probability or 'steady state' of the Markov chain. The outlier value, if *greater* than the other values, effectively serves as an attractor for the particle. In the case above, it's exactly twice as easy to hop *into* this site as it is to hop *out* of it.

Consider a particle moving through a large matrix over a long time. Even at extremely low concentrations of the outlier diffusivity value (e.g., 1 in a million configurations), once the outlier configuration is encountered, and it will eventually be encountered, the particle will be more likely to stay there than to leave, and even if it does leave, it will tend to wander back into that trapping configuration because of its proximity to it. Thus the probability of finding the particle in that state will be higher. In the limit of increasingly large diffusivity, this outlier acts as an absorbing state.

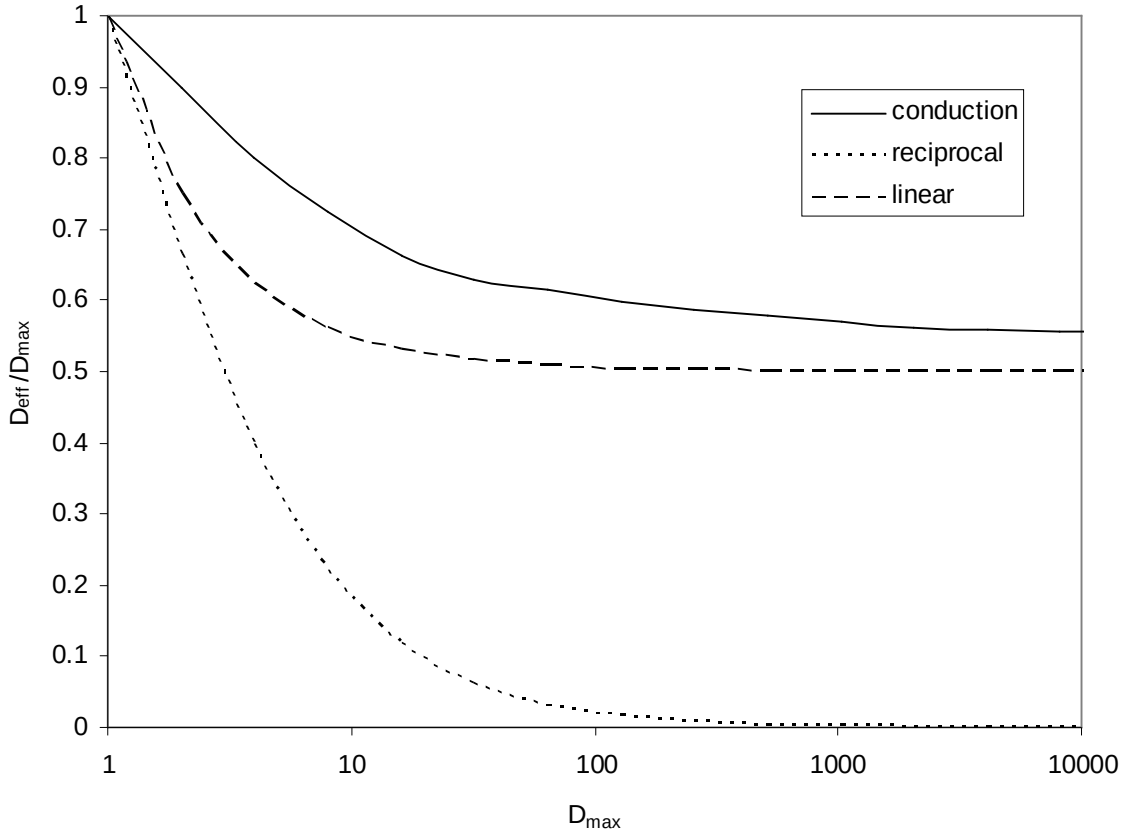


Figure 6-2: The conduction average is consistently higher value than linear and reciprocal averages, because it allows for transport around more insulating elements, as occurs in real materials. Shown is an equal mixture of two types of elements, one with D = 1, the other with D = D_{max}.

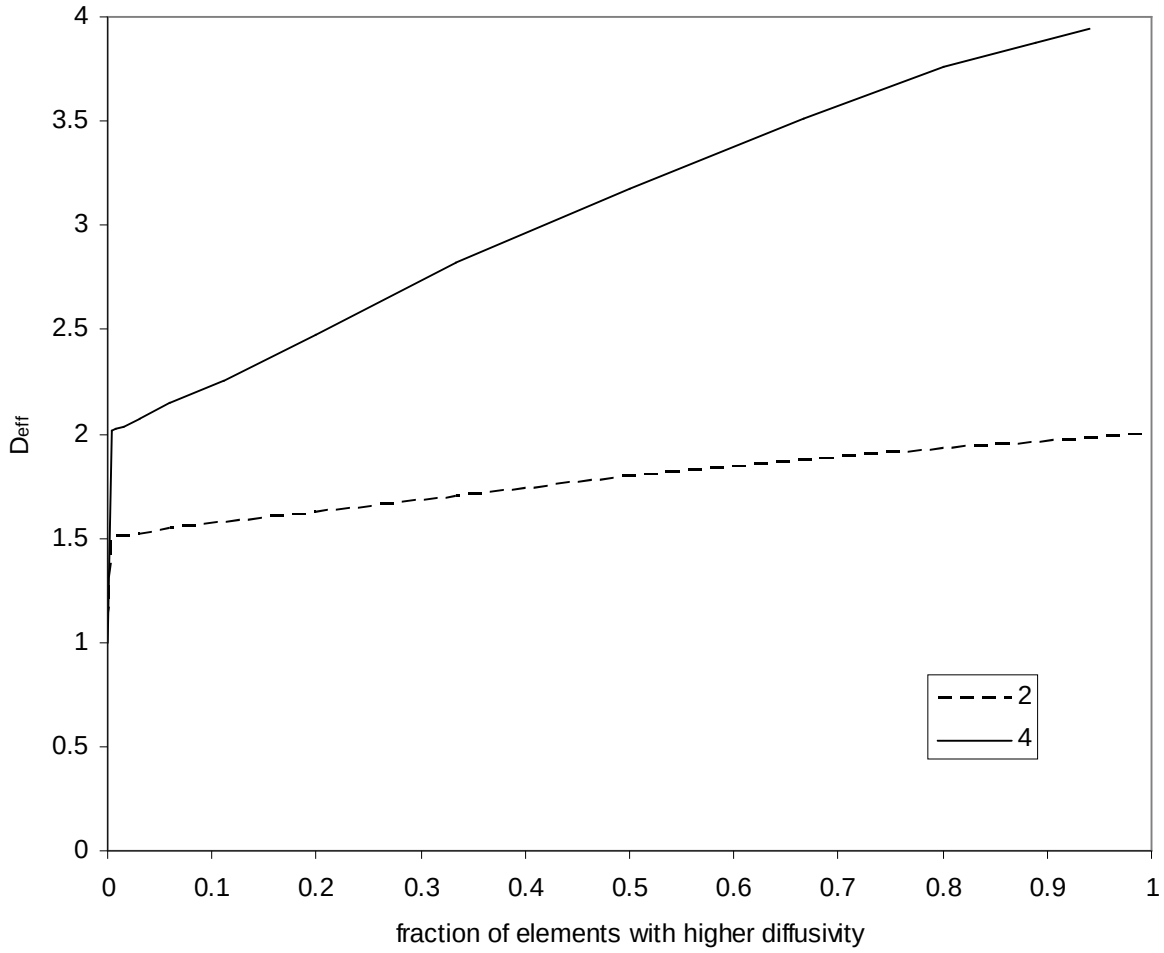


Figure 6-3: Resulting effective diffusivity from a mixture of elements of D = 1, with elements of type D = 2 or D = 4, as shown.

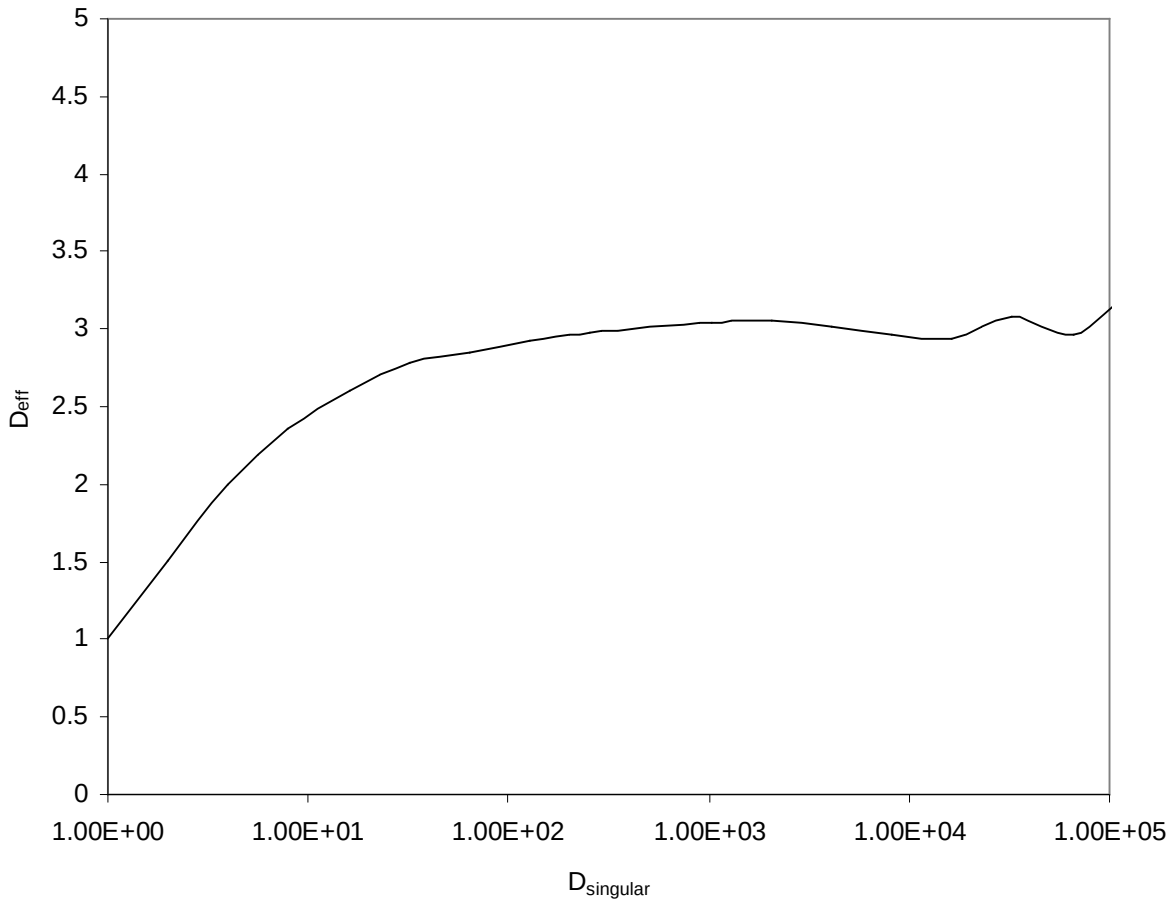


Figure 6-4: Effect of placing an single element of higher diffusivity into a matrix of with all other elements having diffusivity D = 1. The effectivity diffusivity D_{eff} asymptotes to a value of 3. The element occurs with frequency 1ppm, using 10000 samples runs, each for 1000 steps.

PART II: PROBABILISTIC MOLECULAR DYNAMICS

Getting legitimate diffusivity values from traditional molecular dynamics is difficult enough. One must be sure that the polymer has not only had ample opportunity to relax from the tortured initial position in which it was constructed, but also that the penetrant traveling about within it has had ample time to travel well beyond its own diameter in order to escape the anomalous superdiffusive regime which characterizes the startup of such a simulation.

Even then, the diffusivity values obtained can vary by orders of magnitude. For example, a particularly open polymer structure in one of the configurations which can drastically throw off the resultant diffusivity value when an average is taken. A percolating hole in one configuration means essentially no resistance to diffusion at the length scale of the dimensions of the simulation box. This is not at all unphysical, as any material which appears homogenous at a distance will begin to display such inhomogeneity as one zooms in on it. And even when one has overcome these difficulties, there is still the time scale. It is impractical to calculate diffusivities smaller than 10^{-5} cm^2/s by traditional molecular dynamics. It just takes too long.

What is presented is a simulation technique for estimating diffusivity of a penetrant within a given sample configuration. Traditional molecular dynamics was used to generate configurations of the polymeric materials of interest, in the absence of penetrant species. It is assumed that the penetrant does not contribute significantly to the dynamics of the host material over the time scale of diffusion. The penetrants are assumed to exist in exceedingly low concentrations and to move considerably faster than the containing polymer matrix.

The materials of interest were were generated by molecular dynamics using the the amorphous cell module of the commercially available Accelrys Materials Studio[68] software package. The COMPASS[79] force-field (Condensed-phase Optimized Molecular Potentials for

Atomistic Simulation Studies) was used in all simulations. Each configuration was run for 10ps of NVT molecular dynamics at 298 K. Sixty such initial states were created and energy was minimized for 10000 steps, followed by an NVT MD run of 10ps at 298K.

These polymer configurations are used to run PMD simulations for penetrant molecules. As is done for traditional molecular dynamics, a penetrant molecule is introduced into the system configuration and its position is observed as a function of time. The rules for how the particle travels are the signature of the technique. As is observed in traditional molecular dynamics, the penetrant wanders from its insertion point to other void spaces. The dynamics are observed to obey the Einstein relationship[74]:

$$\left\langle R^2 \right\rangle = 6Dt$$

where $\left\langle R^2 \right\rangle$ represents the mean-square distance traveled by the penetrant. By inserting penetrant molecules one at a time and observing their trajectory, a diffusivity is measured for each of the polymer configurations. The conduction average technique is used to obtain the average diffusivity over the complete set of configurations. Experimental values are taken from those compiled by Thran et al[6].

The Technique in Detail:

- The particle is inserted into the system using a modified Metropolis Monte Carlo[59] mechanism: A possible insertion point is selected at random within the sample and an insertion energy is determined. An energy is selected from the Boltzmann distribution. If the insertion energy is less than the energy selected from the distribution, the insertion is accepted. This insures that the particle is inserted into a statistically probable region of the

matrix, avoiding obtuse and irrelevant parts of phase space where the penetrant significantly overlaps the matrix.

- The penetrant particle's velocity is determined by selecting an energy E at random from the Boltzmann distribution: $P(E) = e^{-\beta E}$. This is done with the initial insertion and at every collision point. In this study, only spherically symmetric particles are considered. Allowing for no rotational, vibrational, or electronic excitations, all of the particle's energy is kinetic, and the magnitude of velocity is just $v = \sqrt{2E/m}$. The direction of travel is selected by choosing a point at random on the surface of a sphere.
- The particle travels forward without changing velocity or direction until it collides with the matrix. A collision point is defined as the point along the particle trajectory where its kinetic energy equals the insertion energy for the particle. This is equivalent to pushing the particle forward to the boundary of the insertion continuum as defined by ICM[2].
- Once a collision occurs, a new velocity is selected as above for the particle. Note that even if the new direction and velocity of the particle send it crashing into the matrix, it will immediately reach the collision criterion, forcing the velocity to be resampled until it is no longer crashing into the matrix.
- Time elapsed is calculated by summing the product of the distance traveled for each step of the particle's journey divided by its velocity along that particular trajectory:

$$t = \sum_i l_i v_i$$

RESULTS AND DISCUSSION

Figure 6-5 shows the time series fit as used to evaluate the diffusivity for He in polystyrene. The diffusivity is taken from the slope of this line as:

$$D = \frac{d\left\langle R^2 \right\rangle}{6dt}$$

The time series between the two methods are similar but not identical. Figure 6-6 shows the trajectories for Ne in PSF-M , both methods using the same initial configuration. The deviation is due primarily to statistical noise with some suggestion of a systematic deviation: In many cases, the PMD time series appeared to reach steady state more quickly, but the time series between the two methods were observed to follow a similar long-term trajectory and even to cross frequently.

A variety of polymer/penetrant systems were studied with diffusivities ranging from 10^{-5} to 10^{-9} cm^2/s. In nearly every case studied, there was excellent agreement between probabilistic and traditional molecular dynamics where available (figure 6-7) and also between probabilistic molecular dynamics and experimental data (figure 6-8). Complete tabulated results are shown in tables 6-2 and 6-3.

Notable deviations of PMD results from traditional molecular dynamics results were observed in only one pair of isomers: PMD results for CH4 in 6FDA-6FpDA and 6FDA-6FmDA were shown to overpredict MD diffusivity values by roughly an order of magnitude. The molecular dynamics results shown there are a linear average over configurations. When using the conduction average over the same set of MD configurations, the PMD and MD results are in excellent agreement. However, both models still overpredict experimental diffusivities for the same system by several orders of magnitude. This is attributed to a possible inadequacy of the underlying forcefield model[79, 80] implemented in both types of simulations, in particular as it represents the highly electronegative fluorine atoms in this pair of highly fluorinated isomers. There are known issues with the cross-interactions of fluorocarbon compounds[81, 82], and these isomers are highly fluorinated. Methane is a reasonably polarizable molecule which may be

more attracted to the fluorine than the model predicts. Any resultant decrease, even if very small, in the mobility of methane here can greatly affect the results.

	PMD				MD			
System	# of configurations	# of test particles per configuration	time range for regression (ps)	Box size (Angstroms)	# of configurations	# of test particles per configuration	time step (fs)	time range for regression (ps)
He in PS	10	100	100..1000	25.52520	10	10	1	0..200
CH4 in PS	10	10	200..4000	25.52520	10	5	1	0..200
He in PSF-P	50	20	100..1000	19.8683	10	10	1	10..120
He in PSF-M	50	20	100..1000	19.8130	10	10	1	10..120
He in 6FDA-6FpDA	50	20	100..1000	25.6305	10	10	1	0..120
He in 6FDA-6FmDA	50	20	100..1000	25.46760	10	10	1	0..120
Ne in 6FDA-6FpDA	50	20	100..1000	25.6305	10	10	1	0..120
Ne in 6FDA-6FmDA	50	20	100..1000	25.46760	10	10	1	0..120
Ne in PSF (para)	50	10	20..200	18.1216	10	10	1	20..200
Ne in 3,4'-PSF (meta)	50	10	20..200	18.0731	10	10	1	20..200
Ne in PSF-P	50	10	20..120	19.8683	10	10	1	20..120
Ne in PSF-M	50	10	20..120	19.8130	10	10	1	20..120
CH4 in 6FDA-6FpDA	50	20	0..1000	25.6305	10	5	1	0..200
CH4 in 6FDA-6FmDA	50	20	0..1000	25.46760	10	5	1	0..200
CH4 in PSF (para)	50	20	200..4000	18.1216	10	5		
CH4 in 3,4'-PSF(meta)	50	20	1000..10000	18.0731	10	5		
CH4 in PSF-P				19.8683				
CH4 in PSF-M				19.8130				

Table 6-2: Experimental details for simulations.

System	experiment	MD	MD-CA	PMD	PMD-CA	FFV
He in PS	1.0E-05			1.1E-05	1.7E-05	0.177
CH4 in PS	1.4E-08			8.6E-09	2.2E-08	0.177
He in PSF-P		2.8E-05		1.2E-05	2.2E-05	0.156
He in PSF-M		2.2E-05		1.1E-05	2.1E-05	0.149
He in 6FDA-6FpDA		1.7E-04		9.7E-05	1.4E-04	0.190
He in 6FDA-6FmDA		1.1E-04		9.6E-05	2.0E-04	0.175
Ne in 6FDA-6FpDA		5.3E-05		2.1E-05	3.7E-05	0.190
Ne in 6FDA-6FmDA		3.5E-05		1.9E-05	3.6E-05	0.175
Ne in PSF (para)		3.1E-06	4.3E-06	2.6E-06	5.1E-06	0.156
Ne in 3,4'-PSF (meta)		2.3E-06	3.5E-06	2.0E-06	3.3E-06	0.151
Ne in PSF-P		6.0E-06	2.3E-05	2.8E-06	5.1E-06	0.156
Ne in PSF-M		3.6E-06	1.9E-05	1.8E-06	3.4E-06	0.149
CH4 in 6FDA-6FpDA	6.6E-09	1.2E-06	2.6E-05	8.8E-06	1.6E-05	0.190
CH4 in 6FDA-6FmDA	7.6E-10	8.5E-07	2.2E-05	8.1E-06	1.4E-05	0.175
CH4 in PSF (para)	3.1E-09			4.4E-09	4.9E-09	0.156
CH4 in 3,4'-PSF(meta)	1.2E-09			1.0E-09	2.6E-09	0.151
CH4 in PSF-P	6.0E-09			4.4E-09	5.3E-09	0.156
CH4 in PSF-M	2.8E-09			3.0E-09	4.5E-09	0.149

Table 6-3: Diffusivity results (cm^2/s) of tested penetrant/matrix combinations. Both linear and conduction averages over configurations are shown. Experimental values are from Thran, et al[6].

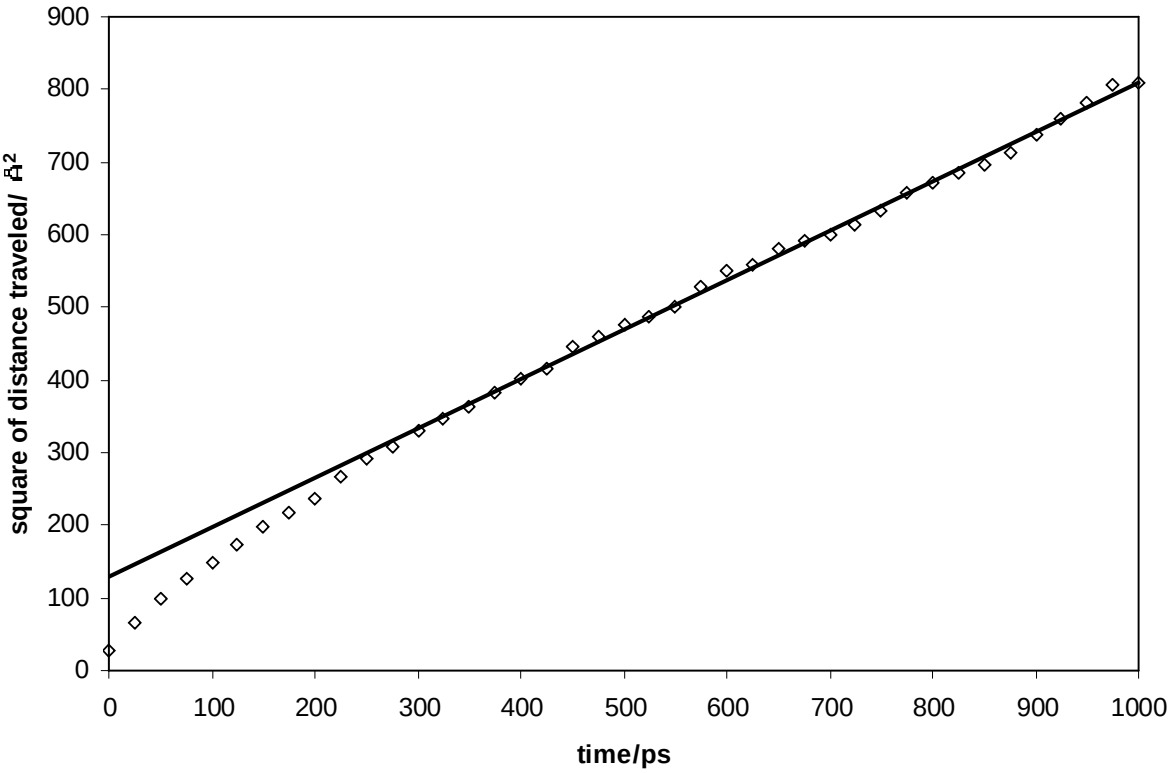


Figure 6-5: Markers show mean-square displacement vs time for He in polystyrene via PMD. Average is over 10 configurations. Straight line is fit to data from 200-1000ps.

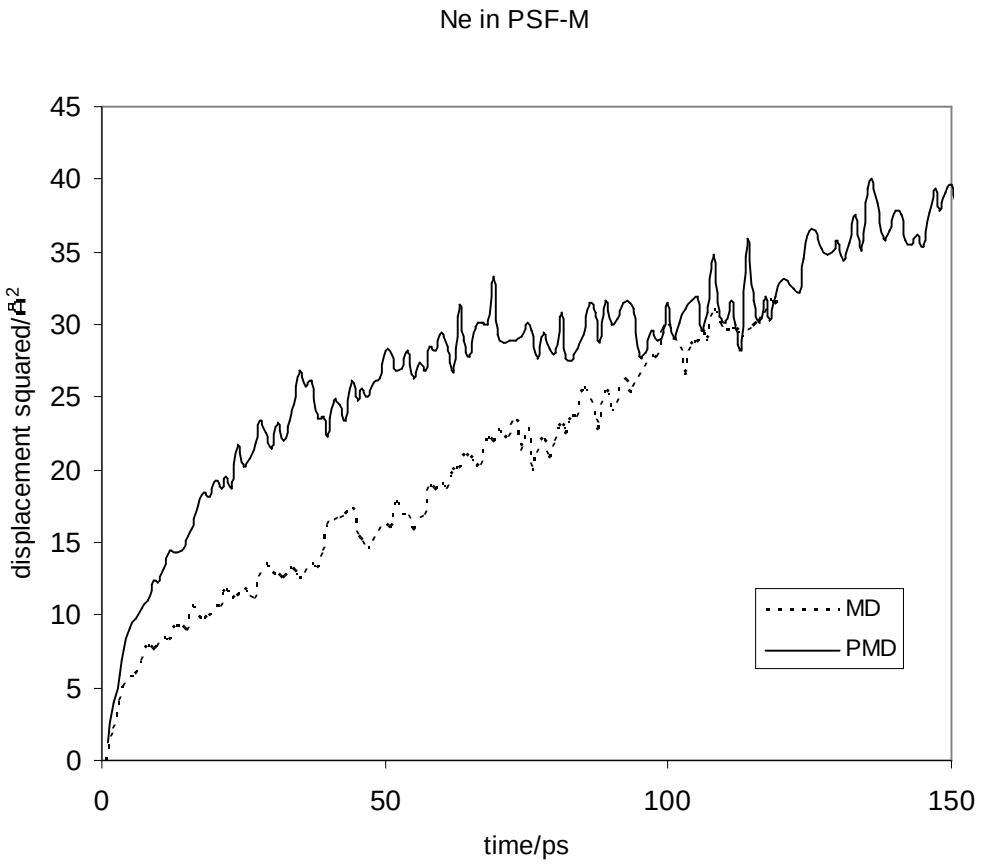


Figure 6-6: Comparative time series for traditional and probabilistic molecular dynamics for Ne in PSF-M.

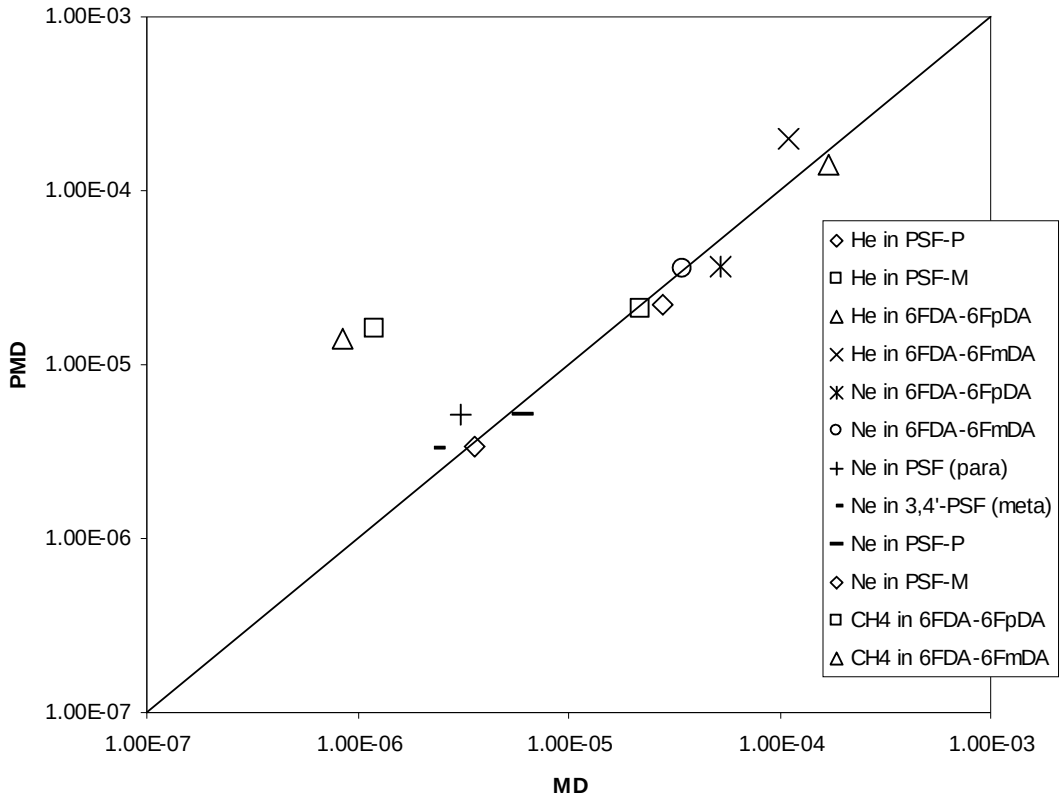


Figure 6-7: Diffusivities obtained by traditional vs probabilistic molecular dynamics in cm2/s. The two outlier points are for methane diffusion in 6FDA-6FpDA and 6FDA-6FmDA. By recomputing the average over configurations for MD using conduction averaging, the values come in line with the PMD values.

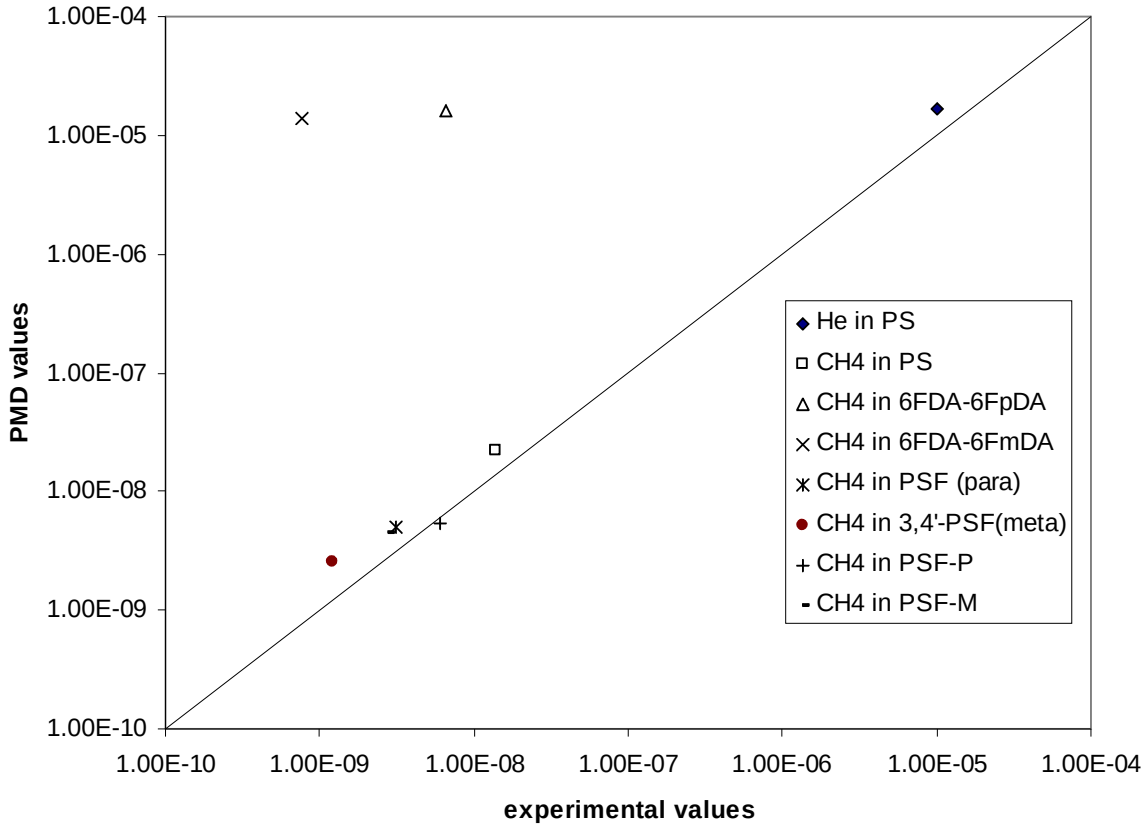


Figure 6-8: Diffusivity values in cm^2/s from PMD vs published experimental values. The two outlier points are for methane diffusion in 6FDA-6FpDA and 6FDA-6FmDA. This is attributed to an inadequacy of the forcefield model for describing the interactions with these highly fluorinated compounds, also observed with traditional molecular dynamics.

CONCLUSIONS

The combination of conduction averaging and probabilistic molecular dynamics extends the range of observable diffusivity well beyond where traditional molecular dynamics data is available due to the time scales needed to obtain results. This is inherently a scaling solution: The diffusion coefficients for the individual configurations are determined by molecular scale simulations. The diffusion coefficients for the overall composite is then determined by a meso-scale stochastic model based on random walk. As an additional caveat, it is suggested that for the conduction average to be justified, the simulation box must be larger than the characteristic length of the voids in the material, which can be measured by CESA[7] or one of its extended methods[2, 8, 34]. This would ensure that what is observed is not a consequence of percolation of free space within any individual configuration, especially in higher free volume polymers.

CHAPTER 7: MISCELLANY

FREE VOLUME PAIR CORRELATION FUNCTION

In describing the relationship of free volume to diffusion, it was considered to look at how free volume is distributed with respect to other free volume. One description of the mechanism of diffusion from the free-volume theory[5, 51, 83] is as the simultaneous probabilities of a molecule having the energy to escape its current surroundings while simultaneously having a free space into which it might jump.

The energy of the molecule follows a simple Arrhenius relationship, with the stickiness of its current position being characterized by a Flory-Huggins type of cross-term. The remaining factor of the existence of a space to jump into is characterized here in terms of a 'free-volume pair correlation function'. Specifically, it is a probability of finding empty space at a given distance from other empty space. In each of the cases, the simulation box is probed by random insertion to find a space that lies outside of the LJ radius of any water or LJ molecule (or the hard sphere radius). Next, a series of 'shells' (corresponding to a fixed set of radii) around this point are probed by random insertions on the surface each shell. These points lie within or outside of the corresponding diameter of the fluid molecules under scrutiny. The probability function, by definition, converges to the unoccupied volume fraction of the fluid. Results are shown for hard spheres, Lennard-Jones 6-12, and SPC/E water, all at a reduced density of 0.75.

Hard spheres show essentially no harmonic information, the possibility of finding unoccupied volume next to unoccupied volume dies off as a critically or overdamped function of radius. This is expected, as there are no attractive forces between hard spheres to cause any meaningful clustering of molecules, and therefore no clustering of free volume. What is

somewhat surprising is that the Lennard-Jones fluid appears to show a statistically significant increase in the clustering of free volume than that the water, in spite of there being lesser attractive forces.

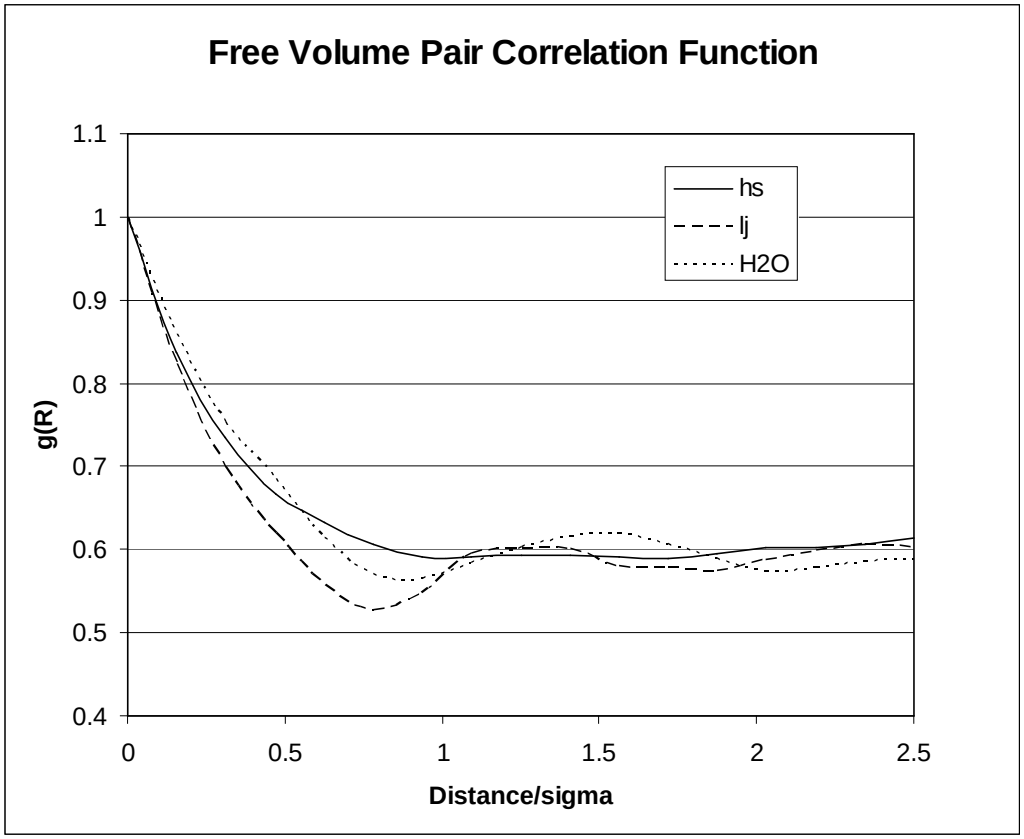


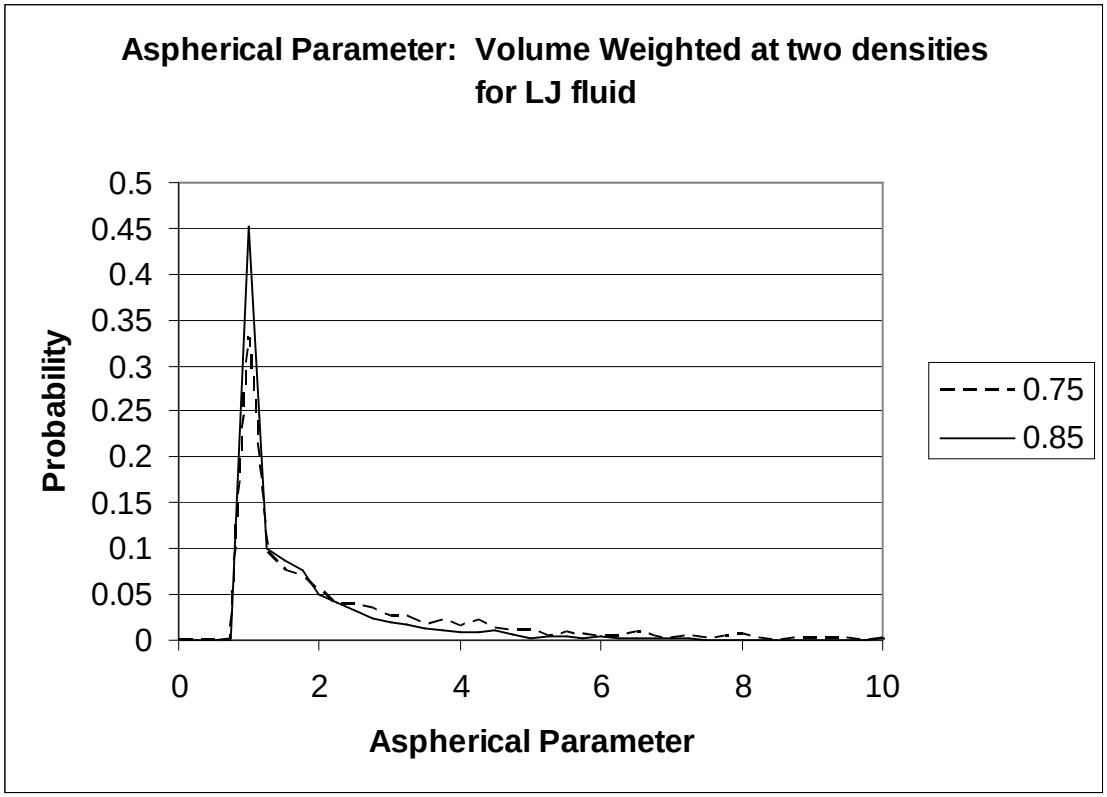
Figure 7-1: Free volume pair correlation function.

ASPHERICAL PARAMETER OF CAVITY CLUSTERS

An attempt was made to characterize the elongation of connected cavity space by defining an 'aspherical' parameter, a combined measure of n appropriate ratio of radius of gyration Rg, surface area s, and volume, independent of the size of the objects:

$$\varphi \equiv \sqrt{\frac{5}{27}} R_g \frac{\sigma}{V}$$

It is identically 1 for a sphere, greater than one for every other shape. The log varies from 0 to positive infinity. It was applied to a Lennard-Jones fluid at 2 densities, with no significant difference in the shapes of the free volume. This may be a result of the fact that the majority of clusters analyzed contained only a single cavity, providing a dominant peak at φ = 1.



CHAPTER 8: RECOMMENDATIONS OF FUTURE WORK

PMD OF DIATOMICS

It may be possible to extend the techniques of probabilistic molecular dynamics to model the transport behavior of diatomic and larger molecules. Due to its symmetry, we were successfully able to model the behavior of methane using a united atom model. For small diatomics such as nitrogen and oxygen, a united atom model may in fact be quite sufficient. For larger or asymmetric molecules, e.g. carbon monoxide, it may suffice to implement an ellipsoidal forcefield for this purpose.

What is poignant and more complicating about larger molecules is the existence of rotational and vibrational quantum states. These states contribute to the heat capacity, but are more elusive, as the degree to which they contribute are a more complicated function of temperature. Translational quantum states contribute beginning at absolute zero temperature, but rotational and vibrational states do not begin to play a significant role until higher temperatures are achieved, and these temperatures are different for different species. As a minimum, good heat capacity data would be needed to develop an adequate model for the motions of these molecules.

If it is possible to model these additional behaviors, it may be possible to develop an appropriate model for 'average' behavior, again reducing the behavior to an analogous united atom model. Alternatively, it may be possible to sample orientational and vibrational states explicitly when moving the diffusant molecule toward a collision point, kind of like throwing hammers at a hole in the wall. It's easier to model a rubber ball than a hammer, but this seems an appropriate analogy.

Carbon dioxide could be a good species to investigate. It is linear and highly symmetric. There is considerable data available thanks to the works of Wang *et al*[8] and Lee.[84] And there is general interest in carbon dioxide separations *via* membrane technology.

CHEMICAL POTENTIAL VIA PMD

There have been discussions about the application of PMD towards determining solubility of a penetrant in a polymer. Solubility is commonly determined by Widom insertions[60] of the penetrant into a set of configurations of polymer. It can also be determined by tracking the potential energy of a penetrant during a molecular dynamics simulation. The second technique is not commonly used because such a run requires extensive computational time for an adequately long molecular dynamics run. Because PMD runs so much faster, this may no longer be as significant an issue.

The second technique involves a temporal monte carlo integration whereas the first involves spatial monte carlo integration. Traditional molecular dynamics is rich with spatial data; PMD is richer in temporal data. PMD uses variable time steps as it has currently been implemented. Such an analysis would require an appropriate weighting of these time steps. The trajectories generated by PMD are inherently less physical than the motions created with traditional molecular dynamics, but overall, there is some promise for the technique, as it does generate considerable information about particle position (and therefore energy) *vs* time to be used in such an analysis, and is not nearly so computationally intensive as traditional molecular dynamics.

APPENDIX A – SOURCE CODE/LIBRARY FUNCTIONS

FTW_STD.H

```
/* ftw_std.h */

#ifndef FTW_STD_INCLUDE
#define FTW_STD_INCLUDE

#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <math.h>

int verbose;

#ifndef V
#define V if (verbose)
#endif

#ifndef TRUE
#define TRUE 1
#endif

#ifndef FALSE
#define FALSE 0
#endif

double getDoubleParameter(char *param, char *value);
int getIntParameter(char *param, char *value);

#endif
```

FTW_STD.C

```
#include "ftw_std.h"

double getDoubleParameter(char *param, char *value)
{
  V printf("%s=%s\n", param, value);
  return strtod(value, NULL);
}

int getIntParameter(char *param, char *value)
{
  V printf("%s=%s\n", param, value);
  return strtol(value, NULL, 10);
}
```

FTW_PARAM.H

```
/* ftw_param.h */
```

```
int command_line_argc;
char **command_line_argv;

void setCommandLineParameters(int argc, char **argv);
void getIntParam(char *param_name, int *parameter);
void getDoubleParam(char *param_name, double *parameter);
void getStringParam(char *param_name, char **parameter);
void getVectorParam(char *param_name, double *parameter1,  double
*parameter2, double *parameter3);
int getFlagParam(char *param_name);
```

FTW_PARAM.C

```
#include <stdlib.h>
#include <string.h>
#include <stdio.h>

//#include "getParameter.h"

int command_line_argc;
char **command_line_argv;

void setCommandLineParameters(int argc, char **argv)
{
  command_line_argc = argc;
  command_line_argv = argv;
}

void getStringParam(char *param_name, char **parameter)
{
  int i=0;

  while (++i<command_line_argc)
  if (strcmp(command_line_argv[i], param_name) == 0)
  {
    if (i+1>=command_line_argc)
    {
      printf("no value specified for %s\n", param_name);
      exit(1);
    }

//    strcpy(parameter, command_line_argv[++i]);
    *parameter = command_line_argv[++i];
  }
}

void getIntParam(char *param_name, int *parameter)
{
  int i=0;

  while (++i<command_line_argc)
  if (strcmp(command_line_argv[i], param_name) == 0)
  {
    if (i+1>=command_line_argc)
    {
```

```
      printf("no value specified for %s\n", param_name);
      exit(1);
    }

    *parameter = (strtol(command_line_argv[++i], NULL, 10));
  }
}

void getDoubleParam(char *param_name, double *parameter)
{
  int i=0;

  while (++i<command_line_argc)
  if (strcmp(command_line_argv[i], param_name) == 0)
  {
    if (i+1>=command_line_argc)
    {
      printf("no value specified for %s\n", param_name);
      exit(1);
    }

    *parameter = (strtod(command_line_argv[++i], NULL));
  }
}

void getVectorParam(char *param_name, double *parameter1,  double
*parameter2, double *parameter3)
{
  int i=0;

  while (++i<command_line_argc)
  if (strcmp(command_line_argv[i], param_name) == 0)
  {
    if (i+3>=command_line_argc)
    {
      printf("not enough values specified for %s\n", param_name);
      exit(1);
    }

    *parameter1 = (strtod(command_line_argv[++i], NULL));
    *parameter2 = (strtod(command_line_argv[++i], NULL));
    *parameter3 = (strtod(command_line_argv[++i], NULL));
  }
}

int getFlagParam(char *param_name)
{
  int i=0;

  while (++i<command_line_argc) if (strcmp(command_line_argv[i], param_name)
== 0) return 1;

  return 0;
}
```

APPENDIX B – FILE FORMATS

.gfg/Generalized conFiGuration file: 5 columns of tab-separated real values, representing x, y, and z coordinates of atoms plus sigma and epsilon parameters.

.cfg/ConFiGuration file: 3 columns of tab-separated real values, representing x, y, and z coordinates.

.hst/HiSTogram file: 2 columns of tab-separated real values

.dst/DiSTribution file: 1 column of real values

APPENDIX C – SOURCE CODE/LENNARD-JONES FLUID IMPLEMENTATION

LJX_MAIN.C

```
/************************************* ljx_main.c
*****************************************/

#include "ljx_main.h"
#include "io_setup.h"
#include "graphics.h"
#include "command_line_parser.h"
#include "energy.h"

#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <ftw_science.h>
#include <ftw_std.h>
#include <ftw_rng.h>

#ifndef MAX_NUMBER_MOLECULES
#define MAX_NUMBER_MOLECULES 16384
#endif

/* non-configurable global params */
double x[MAX_NUMBER_MOLECULES], y[MAX_NUMBER_MOLECULES],
z[MAX_NUMBER_MOLECULES];
int wsize_x, wsize_y, wsize_z;
int change_flag = TRUE; /* signals need to update display */
int monte_carlo_steps = 0;
int monte_carlo_step_counter = 0;

extern int rng_seed;
extern int start_mcs;
extern int end_mcs;
extern int number_of_molecules;
extern int num_pairs;
extern double temperature;
```

```
extern double box_x, box_y, box_z;
extern double target_acceptance_ratio;

extern double pair_list_xoffset[];
extern double pair_list_yoffset[];
extern double pair_list_zoffset[];
extern int pair_list_first[];
extern int pair_list_second[];

double fixed_perturbation_length = .2;
double perturbation_length;
double acceptance_ratio;
int attempted_moves;
int accepted_moves;
int relaxation_allowance = 50;

double dx, dy, dz;
double delta_energy;
int particle_number;

double cutoff_sq = 25.0;
double psi_shift = 1.5999E-05;


int main(int argc, char *argv[])
{
  parseCommandLineOptions(argc, argv);
  initializeRandomNumberGeneratorTo(rng_seed);
  initializeOutput();
  setInitialConditions();
  if (graphicsModeEnabled()) initializeDisplay();

  perturbation_length=fixed_perturbation_length;

  for(monte_carlo_steps=start_mcs; monte_carlo_steps<=end_mcs;
monte_carlo_steps++)
  {
    updatePairList();
    generateOutput();
    attempted_moves = 0;
    accepted_moves = 0;

    for (monte_carlo_step_counter=0;
monte_carlo_step_counter<number_of_molecules; monte_carlo_step_counter++)
    {
      double boltzmann_factor;
      double the_exponential;

      delta_energy = 0;
      attemptMove();
      attempted_moves++;

      if (delta_energy < 0)
      {
        change_flag = 1;
        accepted_moves++;
        continue; /* move accepted */
```

```
      }

      // the following uses reduced temperature
      the_exponential = 0.0 - delta_energy/temperature;
     /* evaluate exponential, unless it's arbitrarily small */
      if (the_exponential > -25)
      {
        boltzmann_factor = exp(the_exponential);
        if (boltzmann_factor > rnd())
        {
          change_flag = 1;
          accepted_moves++;
          continue; /* move accepted */
        }
      }

      // revert move
      x[particle_number] -= dx;
      y[particle_number] -= dy;
      z[particle_number] -= dz;
    }

    if (monte_carlo_steps < relaxation_allowance)
    {
      acceptance_ratio = (0.0 + accepted_moves)/(0.0 + attempted_moves);
      if (acceptance_ratio < target_acceptance_ratio) perturbation_length *=
.9;
      else if (perturbation_length*perturbation_length*perturbation_length*16
< box_x*box_y*box_z) perturbation_length *=1.1;
    }
    else perturbation_length = fixed_perturbation_length;
    if (graphicsModeEnabled() && changeFlagIsSet())
drawGraphicalRepresentation();
  }

  finalizeOutput();
  return 0;
} /* end main */

// Attempt a move, and return the energy change in doing so.
void attemptMove()
{
  int pair_no;

  delta_energy = 0;
  particle_number = floor(number_of_molecules * rnd());

  // calculate energy of all affected pairs before move...
  for (pair_no=0; pair_no<num_pairs; pair_no++)
  {
    if (pair_list_first[pair_no] == particle_number) delta_energy -=
getPairEnergy(pair_no);
    if (pair_list_second[pair_no] == particle_number) delta_energy -=
getPairEnergy(pair_no);
  }

  dx = (rnd() -.5) * perturbation_length;
```

```
  dy = (rnd() -.5) * perturbation_length;
  dz = (rnd() -.5) * perturbation_length;

  x[particle_number] += dx;
  y[particle_number] += dy;
  z[particle_number] += dz;

  // and after the move...
  for (pair_no=0; pair_no<num_pairs; pair_no++)
  {
    if (pair_list_first[pair_no] == particle_number) delta_energy +=
getPairEnergy(pair_no);
    if (pair_list_second[pair_no] == particle_number) delta_energy +=
getPairEnergy(pair_no);
  }

  delta_energy *= 4;
}
```

LJX_MAIN.H

```
/* lj_main.h */

int main();
void perturbSystem();
void updatePosition();
void perturbSystem();
void attemptMove();
```

IO_SETUP.H

```
/* io_setup.h */

void setInitialConditions();
void generateUniqueId();
void finalizeOutput();
void initializeOutput();
void generateOutput();
void loadConfiguration();
```

IO_SETUP.C

```
/* io_setup.c */

#include <ftw_std.h>
#include <ftw_rng.h>
#include <time.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

```
#include <unistd.h>

#include "io_setup.h"
#include "energy.h"

extern char simulation_unique_identifier[];
extern double temperature;
extern int number_of_molecules;
extern int energy_report_frequency;
extern int configuration_threshold;
extern int configuration_frequency;
extern double box_x, box_y, box_z;
extern int monte_carlo_steps;
extern int end_mcs;
extern char* log_file_name;
extern char* output_file_name;
extern char* input_file_name;
extern double x[], y[], z[];

FILE *input_file;

time_t now;

void setInitialConditions()
{
  int i;

  if (input_file_name != NULL) loadConfiguration();
  else
  {
    monte_carlo_steps = 0;
    for (i=0; i<number_of_molecules; i++)
    {
      x[i] = rnd()*box_x;
      y[i] = rnd()*box_y;
      z[i] = rnd()*box_z;
    }
  }
}

void loadConfiguration()
{
  FILE *datastream;
  char line[80];
  char *xs, *ys, *zs;

  number_of_molecules = 0;
  V printf("loading %s...\n", input_file_name);
  datastream = fopen(input_file_name, "r");

  while (1)
  {
    fgets(line, 80, datastream);
    if (feof(datastream)) break;

    xs = strtok(line, "\t");
    ys = strtok(NULL, "\t");
```

```
    zs = strtok(NULL, "\n");

    x[number_of_molecules] = strtod(xs, NULL);
    y[number_of_molecules] = strtod(ys, NULL);
    z[number_of_molecules++] = strtod(zs, NULL);
  }

//  fprintf(stderr,"%d lines read.\n", number_of_molecules);
  fclose(datastream);
}

void generateUniqueId()
{
  int i;
  if (!strcmp(simulation_unique_identifier, "################"))
    for (i=0; i<16; i++) *(simulation_unique_identifier + i) = (char)(rnd() *
26 + 65);
}

/* display headers on output */
void initializeOutput()
{
  now = time(NULL);

  printf("#HT simulation %s started:  %s", simulation_unique_identifier,
ctime(&now));

printf("#HHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHH\n")
;
  printf("#H T=%lf\t N=%d\n", temperature, number_of_molecules);
  printf("#H box dimensions:  \t%lf x %lf x %lf = %lf\n", box_x, box_y,
box_z, box_x*box_y*box_z);
  printf("#H reduced density = %lf\n", (number_of_molecules / (box_x * box_y
* box_z)));
  printf("#H\n");
  printf("#H energy report frequency:\t%d\n", energy_report_frequency);
  printf("#H configuration threshold:\t%d\n", configuration_threshold);
  printf("#H configuration frequency:\t%d\n", configuration_frequency);
  printf("#H run until mcs = %d\n", end_mcs);

printf("#HHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHH\n")
;
  printf("#H\n");
  printf("#H\n");
}

void finalizeOutput()
{
  now = time(NULL);
  printf("#HT simulation %s finished:  %s", simulation_unique_identifier,
ctime(&now));
}

void generateOutput()
{
  int i;
  double system_energy;
```

```
  system_energy = calculateSystemEnergy();
  if (monte_carlo_steps % energy_report_frequency == 0)
printf("#E%06d\t%lf\n", monte_carlo_steps, system_energy);
  if ((monte_carlo_steps > configuration_threshold) && (monte_carlo_steps %
configuration_frequency == 0))
  {
    now = time(NULL);

    printf("#HC%06d\n", monte_carlo_steps);
    printf("#HC%06d dumping configuration at mcs=%d...\n", monte_carlo_steps,
monte_carlo_steps);
    printf("#HC%06d system energy = %lf\n", monte_carlo_steps,
system_energy);
    printf("#HC%06d\n", monte_carlo_steps);
    for (i=0; i<number_of_molecules; i++)
printf("#C%06d\t%d\t%lf\t%lf\t%lf\n", monte_carlo_steps, i, x[i], y[i],
z[i]);
    printf("#HC%06d\n", monte_carlo_steps);
  }
}
```

GRAPHICS.H

```
/* graphics.h */

#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include <X11/Xos.h>
#include <X11/keysym.h>
#include <X11/Xresource.h>
#include <stdlib.h>
#include <stdio.h>
#include <math.h>

void checkForWindowEvent();
void initializeDisplay();
void setChangeFlag();
void resetChangeFlag();
int changeFlagIsSet();
int graphicsModeEnabled();
void drawGraphicalRepresentation();
```

GRAPHICS.C

```
/* graphics.c */

#include "graphics.h"
#include <ftw_science.h>
#include <ftw_std.h>

extern int side_view;
extern int wsize_x, wsize_y, wsize_z;
extern int particle_scale;
```

```
extern double box_x, box_y, box_z;
extern int change_flag;
extern int graphics;
extern int number_of_molecules;
extern int bg_color;
extern int fg_color;
extern int min_color;
extern double x[], y[], z[];

Display *dpy;
Window window, window2;
GC context, context2;
XEvent event;
XGCValues gcvalues;
char *displayname="simulation";
char *display=NULL;
int *z_rank;
int *x_rank;

extern char *display_name_1;
extern char *display_name_2;

int change_flag;

void checkForWindowEvent()
{
  int rstat;

  rstat = XCheckMaskEvent(dpy, ExposureMask | ButtonPressMask, &event);
  if (rstat&&(event.type == ButtonPress))
  {
    XFreeGC(dpy, context);
    XCloseDisplay(dpy);
    exit(0);
  }
}

void initializeDisplay()
{
  wsize_x = (int)(particle_scale * box_x);
  wsize_y = (int)(particle_scale * box_y);
  wsize_z = (int)(particle_scale * box_z);

  z_rank = (int*)malloc(sizeof(int) * number_of_molecules);
  x_rank = (int*)malloc(sizeof(int) * number_of_molecules);

  dpy = XOpenDisplay(display);
  if (dpy == NULL) printf("Can't open the display.\n");

  window = XCreateSimpleWindow(dpy, DefaultRootWindow(dpy), 0, 0, wsize_x,
wsize_y, 0,0,0);
  XSelectInput(dpy, window,
(StructureNotifyMask|ExposureMask|ButtonPressMask|ButtonReleaseMask));
  XMapWindow(dpy, window);
  context = XCreateGC(dpy, window, GCForeground | GCBackground, &gcvalues);
  XStoreName(dpy, window, display_name_1);
  XSetIconName(dpy, window, display_name_2);
```

```
  if (side_view)
  {
    window2 = XCreateSimpleWindow(dpy, DefaultRootWindow(dpy), wsize_x, 0,
wsize_z, wsize_y, 0,0,0);
    XSelectInput(dpy, window2,
(StructureNotifyMask|ExposureMask|ButtonPressMask|ButtonReleaseMask));
    XMapWindow(dpy, window2);
    context2 = XCreateGC(dpy, window2, GCForeground | GCBackground,
&gcvalues);
    XStoreName(dpy, window2, display_name_2);
    XSetIconName(dpy, window2, display_name_2);
  }

  checkForWindowEvent();
}

void setChangeFlag()
{
  change_flag = TRUE;
}

void resetChangeFlag()
{
  change_flag = FALSE;
}

int changeFlagIsSet()
{
  return change_flag;
}

int graphicsModeEnabled()
{
  return graphics;
}

void drawGraphicalRepresentation()
{
  int xx, yy, zz;
  int i, j;
  int z_shade, x_shade;
  int temporary;

  if (!changeFlagIsSet()) return;

  for (i=0; i<number_of_molecules; i++) z_rank[i] = i;
  gcvalues.foreground = bg_color;
  XChangeGC(dpy, context, GCForeground, &gcvalues);
  XFillRectangle(dpy, window, context, 0, 0, wsize_x*2, wsize_y*2);

  /* order by z values */
  for (i=number_of_molecules; i>0; i--)
    for (j=0; j<i-1; j++)
      if (z[z_rank[j]] < z[z_rank[j+1]])
      {
        temporary = z_rank[j];
```

```
        z_rank[j] = z_rank[j+1];
        z_rank[j+1] = temporary;
      }

  for(i=0;i<number_of_molecules;i++)
  {
    xx=floor(x[z_rank[i]]*particle_scale - particle_scale/2);
    yy=floor(y[z_rank[i]]*particle_scale - particle_scale/2);
    z_shade = fg_color - floor((fg_color - min_color) * z[z_rank[i]] /
box_z);
    gcvalues.foreground = z_shade;
    XChangeGC(dpy, context, GCForeground, &gcvalues);
    XFillArc(dpy, window, context, xx, yy, particle_scale, particle_scale, 0,
360*64);
    XFillArc(dpy, window, context, xx, yy + wsize_y, particle_scale,
particle_scale, 0, 360*64);
    XFillArc(dpy, window, context, xx + wsize_x, yy, particle_scale,
particle_scale, 0, 360*64);
    XFillArc(dpy, window, context, xx + wsize_x, yy + wsize_y,
particle_scale, particle_scale, 0, 360*64);
  }

  if (side_view)
  {
    for (i=0; i<number_of_molecules; i++) x_rank[i] = i;
    gcvalues.foreground = bg_color;
    XChangeGC(dpy, context2, GCForeground, &gcvalues);
    XFillRectangle(dpy, window2, context2, -wsize_z/2, -wsize_y/2, wsize_z*2,
wsize_y*2);

    /* order by x values */
    for (i=number_of_molecules; i>0; i--)
      for (j=0; j<i-1; j++)
        if (x[x_rank[j]] > x[x_rank[j+1]])
        {
          temporary = x_rank[j];
          x_rank[j] = x_rank[j+1];
          x_rank[j+1] = temporary;
        }

    for(i=0; i<number_of_molecules;i++)
    {
      x_shade = min_color + floor((fg_color - min_color) * x[x_rank[i]] /
box_x);
      yy=floor(y[x_rank[i]]*particle_scale - particle_scale/2);
      zz=floor(z[x_rank[i]]*particle_scale - particle_scale/2);
      gcvalues.foreground = x_shade;
      XChangeGC(dpy, context2, GCForeground, &gcvalues);
      XFillArc(dpy, window2, context2, zz, yy, particle_scale,
particle_scale, 0, 360*64);
      XFillArc(dpy, window2, context2, zz, yy + wsize_y, particle_scale,
particle_scale, 0, 360*64);
      XFillArc(dpy, window2, context2, wsize_z + zz, yy, particle_scale,
particle_scale, 0, 360*64);
      XFillArc(dpy, window2, context2, wsize_z + zz, yy + wsize_y,
particle_scale, particle_scale, 0, 360*64);
    }
```

```
  }

  XFlush(dpy);
  checkForWindowEvent();
  resetChangeFlag();
}
```

COMMAND_LINE_PARSER.C

```
/* command_line_parser.c */

#include "ftw_std.h"
#include "ftw_rng.h"
//#include "command_line_parser.h"

extern int verbose;

FILE *instream;
double box_x=7, box_y=7, box_z=7;
int mirror_depth = 1;
double sfactor = 1.0;

char *prog_name;

void parseCommandLineOptions(int argc, char *argv[])
{
  int i=0;

  prog_name = argv[0];
  if (argc < 2) printUsage();
  instream = stdin;

  while (++i<argc)
  {
    if ((*argv[i]) != '-') instream = fopen(argv[i], "r");
    else if (!strcmp(argv[i], "-usage")) printUsage();
    else if (!strcmp(argv[i], "-v")) verbose = TRUE;
    else if (!strcmp(argv[i], "-mirror_depth")) mirror_depth =
getIntParameter("mirror_depth", argv[++i]);
    else if (!strcmp(argv[i], "-sfactor")) sfactor =
getIntParameter("sfactor", argv[++i]);
    else if (!strcmp(argv[i], "-randomize")) randomize();
    else if (!strcmp(argv[i], "-box"))
    {
      box_x = getDoubleParameter("box_x", argv[++i]);
      box_y = getDoubleParameter("box_y", argv[++i]);
      box_z = getDoubleParameter("box_z", argv[++i]);
    }
  }
}

printUsage()
{
  printf("usage:  %s [-options]\n", prog_name);
  exit(0);
```

```
}
```

ENERGY.H

```
/* energy.h */

double calculateSystemEnergy();
void updatePairList();
double getPairEnergy(int pair_no);
```

ENERGY.C

```
/* energy.c */

#include "energy.h"

#define NPAIRS 1048576

extern double x[], y[], z[];
extern double box_x, box_y, box_z;
extern int number_of_molecules;
extern int monte_carlo_steps;
extern int relaxation_allowance;
int num_pairs;

int pair_list_first[NPAIRS];
int pair_list_second[NPAIRS];
double pair_list_xoffset[NPAIRS];
double pair_list_yoffset[NPAIRS];
double pair_list_zoffset[NPAIRS];

int time_to_update=0;

// program uses values of sigma=1,epsilon=1
// psi_shift is the shift-and-truncate correction.
extern double psi_shift;
extern double cutoff_sq;

/* returns energy per mole of LJ centers */
double calculateSystemEnergy()
{
  double energy = 0;
  int pair_no;

  // if (monte_carlo_steps < relaxation_allowance) return
calculateSystemEnergyRigorously();

  // now evaluate energies
  for (pair_no=0; pair_no<num_pairs; pair_no++) energy +=
getPairEnergy(pair_no);

  return 4*energy;
}
```

```
double getPairEnergy(int pair_no)
{
  double dx, dy, dz;
  double energy=0;
  double r_sq, r6, r12;

  dx = x[pair_list_second[pair_no]] + pair_list_xoffset[pair_no] -
x[pair_list_first[pair_no]];
  dy = y[pair_list_second[pair_no]] + pair_list_yoffset[pair_no] -
y[pair_list_first[pair_no]];
  dz = z[pair_list_second[pair_no]] + pair_list_zoffset[pair_no] -
z[pair_list_first[pair_no]];
  r_sq = dx*dx + dy*dy + dz*dz;
  if (r_sq < cutoff_sq)
  {
    r6 = r_sq * r_sq * r_sq;
    r12 = r6 * r6;
    energy = (1/r12 - 1/r6 + psi_shift);
  }

  return energy;
}

void updatePairList()
{
  int i, j;
  int shift_x, shift_y, shift_z;
  int close_x=0, close_y=0, close_z=0;
  double dsq, min_dsq;
  double dx, dy, dz;

  // correctForBoundaries
  for (i=0; i<number_of_molecules; i++)
  {
    if (x[i] >= box_x) x[i] -= box_x;
    if (y[i] >= box_y) y[i] -= box_y;
    if (z[i] >= box_z) z[i] -= box_z;

    if (x[i] < 0) x[i] += box_x;
    if (y[i] < 0) y[i] += box_y;
    if (z[i] < 0) z[i] += box_z;
  }

  num_pairs = 0;
  for (i=0; i<number_of_molecules - 1; i++)
  for (j=i + 1; j<number_of_molecules; j++)
  {
//    min_dsq = 10;  // set an initial value
    min_dsq = cutoff_sq*1.3;  // set an initial value
    for (shift_x = -1; shift_x<=1; shift_x++)
    for (shift_y = -1; shift_y<=1; shift_y++)
    for (shift_z = -1; shift_z<=1; shift_z++)
    {
      dx = shift_x * box_x + x[j] - x[i];
      dy = shift_y * box_y + y[j] - y[i];
      dz = shift_z * box_z + z[j] - z[i];
      dsq = dx*dx + dy*dy + dz*dz;
```

```
      if (dsq < min_dsq)
      {
        close_x = shift_x;
        close_y = shift_y;
        close_z = shift_z;
        min_dsq = dsq;
      }
    }

//    if (min_dsq < 9) // add it to pair list
    if (min_dsq < cutoff_sq*1.25) // add it to pair list
    {
      pair_list_first[num_pairs] = i;
      pair_list_second[num_pairs] = j;
      pair_list_xoffset[num_pairs] = close_x * box_x;
      pair_list_yoffset[num_pairs] = close_y * box_y;
      pair_list_zoffset[num_pairs] = close_z * box_z;
      num_pairs++;
    }
  } // next j, i

  time_to_update = number_of_molecules * 10;
}
```

APPENDIX D – SOURCE CODE/PMD IMPLEMENTATION

UTRWT69.C

This program implements PMD using a Lennard-Jones 6-9 potential form in conjunction with COMPASS[79, 80] forcefield parameters. The program accepts a generalized configuration file (.gfg) as input, and will return two columns of output: time in nanoseconds and displacement in Angstroms^2 if the –time_series flag is specified on the command line. Penetrant properties are specified on the command line.

```
/* utrwt69.c */

void readConfiguration();
double calculateEnergy();

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <math.h>
#include <ftw_std.h>
#include <ftw_param.h>
#include <time.h>

#define MAX_NUM_MOLECULES 16384
```

```
#define MAX_CLOSE 2048
#define TIME_BINS 20400
#define PI 3.141592653589796264

double x[MAX_NUM_MOLECULES];
double y[MAX_NUM_MOLECULES];
double z[MAX_NUM_MOLECULES];
double sigma[MAX_NUM_MOLECULES];
double epsilon[MAX_NUM_MOLECULES];

double close_x[MAX_CLOSE], close_y[MAX_CLOSE], close_z[MAX_CLOSE];
double close_sigma[MAX_CLOSE];
double close_epsilon[MAX_CLOSE];

double box_x=6, box_y=6, box_z=6;
double verlet_cutoff=100.0;
double time_series_R_delta_t[TIME_BINS];
double time_series_R_sq_delta_t[TIME_BINS];
double time_series_delta_t[TIME_BINS];

double step_size = 0.10;

double T=298;
double energy, new_energy;

int detail=0;
int gradient=0;
int number_of_samples = 1;
int time_series=0;

double test_x0, test_y0, test_z0;
double test_x, test_y, test_z;
double collision_x, collision_y, collision_z;
double verlet_center_x, verlet_center_y, verlet_center_z;

// Default particle is Helium
double test_diameter = 2.9; // Angstroms
double test_epsilon = 2.52; // in K
double test_mass = 0.004; // kg/mol
double target_time = 100.0; // picoseconds
double threshold_time = 10.0; // picoseconds

int seed = 123450;
int successes;

int bin_size=1;
int report_frequency=10;
int number_of_steps = 1024;
int number_of_molecules = 0;
int close_molecules;
double drift_x, drift_y, drift_z;

/* NOTE:  sigma is specified in Angstroms, epsilon in K, T in K */
const double rand_step = 1.0/RAND_MAX;

int main(int argc, char *argv[])
{
```

```
  double sq_distance_from_initial_pt; //
  double n_steps_taken; // how many steps from initial insertion
  double dx, dy, dz, dd, dt, d; // distance deltas...
  double phi, theta; // used for choosing direction
  double x_step, y_step, z_step; // step size in that direction
  double mid_x, mid_y, mid_z, mid_t;
  double R_t;
  int t_bin;
  double time_elapsed;
  double collision_t;
  double kinetic_energy;
  double old_energy, new_energy;
  double velocity;
  double time_step;
  double D;
  int bisections;
  int i;
  double intercollision_distance;
  double bisection_factor;
  double grad_x, grad_y, grad_z, grad;

  setCommandLineParameters(argc, argv);
  verbose = getFlagParam("-verbose");
  getIntParam("-seed", &seed);
  if (getFlagParam("-randomize")) seed=time(NULL);
  srand(seed);
  if (getFlagParam("-usage"))
  {
    printf("\nusage:  configuration in, list of path lengths  and times
out\n\n");
    printf("        -box [ 6.0 6.0 6.0 ] (Angstroms)\n");
    printf("        -test_diameter [ 2.9 ] (Angstroms)\n");
    printf("        -test_epsilon [ 2.52 ] (K)\n");
    printf("        -test_mass [ 0.004 ] (kg)\n");
    printf("        -T [ 298.0 ]\n");
    printf("        -n [ 1 ]\n");
    printf("        -N [ 1024 ]\n");
    printf("        -step_size [ .100 ]\n");
    printf("        -verlet_cutoff [ 100.0 ]\n");
    printf("        -target_time (in picoseconds) [ 100.0 ]\n");
    printf("        -threshold_time (in picoseconds) [ 10.0 ]\n");
    printf("        -report_frequency (in picoseconds) [ 10 ]\n");
    printf("        -bin_size(in picoseconds) [ 1 ]\n");
    printf("        -seed [ 123450 ]\n");
    printf("        -detail\n");
    printf("        -time_series\n");
    printf("        -gradient\n");
    printf("        -randomize\n\n");

    exit(0);
  }

  srand(seed);

  getVectorParam("-box", &box_x, &box_y, &box_z);
  getDoubleParam("-step_size", &step_size);
  getDoubleParam("-verlet_cutoff", &verlet_cutoff);
```

```
getDoubleParam("-test_diameter", &test_diameter);
getDoubleParam("-target_time", &target_time);
target_time*=1.0e-12;
getDoubleParam("-threshold_time", &threshold_time);
threshold_time*=1.0e-12;
getDoubleParam("-test_epsilon", &test_epsilon);
getDoubleParam("-test_mass", &test_mass);
getDoubleParam("-T", &T);
getIntParam("-n", &number_of_samples);
getIntParam("-N", &number_of_steps);
getIntParam("-bin_size", &bin_size);
detail = getFlagParam("-detail");
time_series = getFlagParam("-time_series");
gradient = getFlagParam("-gradient");

readConfiguration();

for (i=0; i<1000; i++)
{
  time_series_R_delta_t[i] = 0;
  time_series_R_sq_delta_t[i] = 0;
  time_series_delta_t[i] = 0;
}

// loop over # of insertions
while (number_of_samples-- > 0)
{
  generateTestPoint();
  new_energy = old_energy = calculateEnergy();
  drift_x=drift_y=drift_z=0;
  time_elapsed=0;

  while (time_elapsed<target_time)
  {
    if (gradient)
    {
      // direction of gradient
      test_x+=.000001;
      grad_x=calculateEnergy();
      test_x-=.000002;
      grad_x-=calculateEnergy();
      test_x+=.000001;

      test_y+=.000001;
      grad_y=calculateEnergy();
      test_y-=.000002;
      grad_y-=calculateEnergy();
      test_y+=.000001;

      test_z+=.000001;
      grad_z=calculateEnergy();
      test_z-=.000002;
      grad_z-=calculateEnergy();
      test_z+=.000001;

      grad=sqrt(grad_x*grad_x + grad_y*grad_y + grad_z*grad_z);
      x_step=-step_size*grad_x/grad;
```

```
        y_step=-step_size*grad_y/grad;
        z_step=-step_size*grad_z/grad;
      }
      else
      {
        // pick a direction
        phi = 2*PI*rand()*rand_step;
        theta = PI*rand()*rand_step;
        x_step = step_size*cos(phi)*sin(theta);
        y_step = step_size*sin(phi)*sin(theta);
        z_step = step_size*cos(theta);
      }

      // pick a velocity
      kinetic_energy = -1.5*log(rand_step*rand())*8.314*T; // in J/mol
      velocity = sqrt(2.0*kinetic_energy/test_mass) * 1.0e+10; // in A/s
      collision_x = test_x;
      collision_y = test_y;
      collision_z = test_z;
      collision_t = time_elapsed;
      intercollision_distance=0;
      bisection_factor=1.0;

      // extend ray from collision point
      for (bisections=0; bisections<=15; )
      {
        test_x += x_step;
        test_y += y_step;
        test_z += z_step;

        dx=test_x - verlet_center_x;
        dy=test_y - verlet_center_y;
        dz=test_z - verlet_center_z;

        if (dx*dx + dy*dy + dz*dz > .01 * verlet_cutoff) makeVerletList();

        old_energy = new_energy;
        new_energy = calculateEnergy();
        if ((new_energy > (kinetic_energy / 8.314)) && (new_energy >=
old_energy))
        {
          new_energy = old_energy;

          test_x-=x_step;
          test_y-=y_step;
          test_z-=z_step;

          x_step*=.5;
          y_step*=.5;
          z_step*=.5;
          bisection_factor*=.5;

          bisections++;
        }

        // figure out what time it is
        intercollision_distance+=step_size*bisection_factor;
```

```
        time_elapsed += step_size*bisection_factor/velocity;
        if (detail && (time_elapsed > threshold_time))
        {
          dx = test_x - test_x0 + drift_x;
          dy = test_y - test_y0 + drift_y;
          dz = test_z - test_z0 + drift_z;
          dd = dx*dx + dy*dy + dz*dz;
          d = sqrt(dd);
          D=1.0e-16*dd/(6*time_elapsed);
          printf("%1.12lf\t%6.0lf\t%1.12lf\t%1.12lf\t%1.12lf\t%1.12lf\n", d,
1.0e+12*time_elapsed, D, test_x, test_y, test_z);
        }
      } // end loop over bisections

      if (time_series)
      {
        mid_x = (test_x+collision_x) * .5;
        mid_y = (test_y+collision_y) * .5;
        mid_z = (test_z+collision_z) * .5;
        dx=mid_x-test_x0+drift_x;
        dy=mid_y-test_y0+drift_y;
        dz=mid_z-test_z0+drift_z;
        R_t = sqrt(dx*dx + dy*dy + dz*dz);
        dt = time_elapsed - collision_t;
        mid_t = collision_t + (dt * .5);

        t_bin = floor(1.0e+12*mid_t/bin_size + .5);

        // t_bin = floor(1000000000000*mid_t + .5);

        time_series_R_delta_t[t_bin] += R_t*dt;
        time_series_R_sq_delta_t[t_bin] += R_t*R_t*dt;
        time_series_delta_t[t_bin]+= dt;
      }

    } // end loop until target_time

    // total distance traveled for this insertion
    dx = test_x - test_x0 + drift_x;
    dy = test_y - test_y0 + drift_y;
    dz = test_z - test_z0 + drift_z;
    dd = dx*dx + dy*dy + dz*dz;
    d = sqrt(dd);

  } // end loop over all samples

  if (time_series) for (i=0; i*bin_size < (time_elapsed * 1.0e+12); i++)
    printf("%d\t%lf\n", i*bin_size,
time_series_R_sq_delta_t[i]/time_series_delta_t[i]);

  return 0;
}

generateTestPoint()
{
  test_x = test_x0 = rand() * rand_step * box_x;
  test_y = test_y0 = rand() * rand_step * box_y;
```

```
  test_z = test_z0 = rand() * rand_step * box_z;

  makeVerletList();

  while (rand() * rand_step > exp(-(calculateEnergy()/T)))
  {
    test_x = test_x0 = rand() * rand_step * box_x;
    test_y = test_y0 = rand() * rand_step * box_y;
    test_z = test_z0 = rand() * rand_step * box_z;

    makeVerletList();
  }
}

makeVerletList()
{
  int i;
  double dx, dy, dz, dd;
  double shift_x, shift_y, shift_z;
  double close_sigma6;

  while (test_x > box_x)
  {
    test_x -= box_x;
    collision_x -= box_x;
    drift_x += box_x;
  }
  while (test_y > box_y)
  {
    test_y -= box_y;
    collision_y -= box_y;
    drift_y += box_y;
  }
  while (test_z > box_z)
  {
    test_z -= box_z;
    collision_z -= box_z;
    drift_z += box_z;
  }

  while (test_x < 0)
  {
    test_x += box_x;
    collision_x += box_x;
    drift_x -= box_x;
  }
  while (test_y < 0)
  {
    test_y += box_y;
    collision_y += box_y;
    drift_y -= box_y;
  }
  while (test_z < 0)
  {
    test_z += box_z;
    collision_z += box_z;
    drift_z -= box_z;
```

```
  }

  verlet_center_x=test_x;
  verlet_center_y=test_y;
  verlet_center_z=test_z;

  close_molecules=0;

  for (i=0; i<number_of_molecules; i++)
  {
    for (shift_x = -box_x; shift_x <= box_x; shift_x += box_x)
    for (shift_y = -box_y; shift_y <= box_y; shift_y += box_y)
    for (shift_z = -box_z; shift_z <= box_z; shift_z += box_z)
    {
      dx = shift_x + x[i] - test_x;
      dy = shift_y + y[i] - test_y;
      dz = shift_z + z[i] - test_z;

      dd = dx*dx + dy*dy + dz*dz;

      if (dd < verlet_cutoff)
      {
        close_x[close_molecules] = shift_x + x[i];
        close_y[close_molecules] = shift_y + y[i];
        close_z[close_molecules] = shift_z + z[i];

//
close_sigma6=((sigma[i]*sigma[i]*sigma[i]*sigma[i]*sigma[i]*sigma[i])
+(test_diameter*test_diameter*test_diameter*test_diameter*test_diameter*test_
diameter))*.5;
//        close_sigma[close_molecules] = pow(close_sigma6,
.16666666666666666666666666666);
//        close_epsilon[close_molecules] = sqrt(test_epsilon * epsilon[i]) *
(test_diameter * sigma[i] * test_diameter * sigma[i] * test_diameter *
sigma[i]) / close_sigma6;

        close_sigma[close_molecules] = sigma[i];
        close_epsilon[close_molecules] = epsilon[i];

        close_molecules++;
      }
    }
  }
}

// returns the energy change by insertion of a test particle at test_x,
test_y, test_z
double calculateEnergy()
{
  double repulsion=0;
  double attraction=0;
  double alpha;
  double dx, dy, dz, d, dd;
  int i;

  for (i=0; i<close_molecules; i++)
  {
```

```
    dx = close_x[i] - test_x;
    dy = close_y[i] - test_y;
    dz = close_z[i] - test_z;
    dd = dx*dx + dy*dy + dz*dz;

    d=sqrt(dd);
    alpha = close_sigma[i]*close_sigma[i]*close_sigma[i]/(d*dd);

    repulsion += close_epsilon[i] * alpha*alpha*alpha;
    attraction += close_epsilon[i] * alpha*alpha;
  }

  return (2*repulsion - 3*attraction);
}

void readConfiguration()
{
  char line[80];
  char *xs, *ys, *zs;
  char *sigmas, *epsilons;
  double sigmad, epsilond, sigma6;

  number_of_molecules = 0;

  while (1)
  {
    fgets(line, 80, stdin);
    if (feof(stdin)) break;

    xs = strtok(line, "\t");
    ys = strtok(NULL, "\t");
    zs = strtok(NULL, "\t");
    sigmas = strtok(NULL, "\t");
    epsilons = strtok(NULL, "\n");

    x[number_of_molecules] = strtod(xs, NULL);
    y[number_of_molecules] = strtod(ys, NULL);
    z[number_of_molecules] = strtod(zs, NULL);
    //sigma[number_of_molecules] = strtod(sigmas, NULL);
    sigmad = strtod(sigmas, NULL);
    sigma6 =
((sigmad*sigmad*sigmad*sigmad*sigmad*sigmad)+(test_diameter*test_diameter*tes
t_diameter*test_diameter*test_diameter*test_diameter))*.5;
    sigma[number_of_molecules] = pow(sigma6, .16666666666666666666666666666);
    epsilond = strtod(epsilons, NULL);
    epsilon[number_of_molecules] =
sqrt(test_epsilon*epsilond)*(test_diameter*sigmad*test_diameter*sigmad*test_d
iameter*sigmad)/sigma6;

    number_of_molecules++;
  }

  V printf("%d lines read.\n", number_of_molecules);
  fclose(stdin);
}
```

APPENDIX E – SOURCE CODE/CONDUCTION AVERAGE IMPLEMENTATION

MATRIX.C

```
/* matrix.c */

/* input:  .dst */
/* output:  singular value (average of all entries, based on diffusion in 3-D
cubic matrix) */

#include <ftw_std.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <math.h>

double the_matrix[256][256][256];
double input_vals[1000000];
int n_inputs = 0;
const double rand_step = 1.0/RAND_MAX;
double Dmax=0;
int n_attempts=10000;
int n_steps=1000;

int i,j,k;
int i_0,j_0,k_0;
int new_i,new_j,new_k;

int main(int argc, char *argv[])
{
  char line[80];
  char *xs;
  double D;
  int attempts, steps;
  int direction;
  double distance_sq=0,sum_distance_sq=0;

  while (TRUE)
  {
    fgets(line, 80, stdin);
    if (feof(stdin)) break;

    xs = strtok(line, "\n");
    D=strtod(xs, NULL);
    if (D>Dmax) Dmax=D;
    input_vals[n_inputs++] = D;
  }
//printf("%d\n", n_inputs);

  for(i=0;i<256;i++)
  for(j=0;j<256;j++)
  for(k=0;k<256;k++)
    the_matrix[i][j][k] = input_vals[(int)(rand()*rand_step*n_inputs)];

  // now start the insertions and wanderers
```

```
  for (attempts=0;attempts<n_attempts;attempts++)
  {
    // pick an insertion point
    new_i=i=i_0=65536+(rand()&255);
    new_j=j=j_0=65536+(rand()&255);
    new_k=k=k_0=65536+(rand()&255);

    for (steps=0;steps<n_steps;steps++)
    {
      // pick a direction
      direction=floor(rand()*rand_step*6);
      switch (direction)
      {
        case 0:
          new_i = i-1;
          break;
        case 1:
          new_i = i+1;
          break;
        case 2:
          new_j = j-1;
          break;
        case 3:
          new_j = j+1;
          break;
        case 4:
          new_k = k-1;
          break;
        case 5:
          new_k = k+1;
          break;
        default:
          printf("direction=%d, wtf?\n",direction);
          exit(0);
      }

      // accept/reject move
      if
((rand()*rand_step*Dmax)<the_matrix[new_i&255][new_j&255][new_k&255]) {
i=new_i; j=new_j; k=new_k; }
//printf("%d\t%d\t%d\n",i,j,k);
    }

    // see how far we've traveled...
    distance_sq=(i-i_0)*(i-i_0)+(j-j_0)*(j-j_0)+(k-k_0)*(k-k_0);
    // more code here to keep track of set of distances...
//printf("%lf\n", distance_sq);
    sum_distance_sq += distance_sq;
  }

  // code to translate the set of distances back into a diffusion
coefficeinet
printf("%12.24lf\n", Dmax*sum_distance_sq/(n_attempts*n_steps));

}
```

APPENDIX F – SOURCE CODE/LENNARD-JONES FLUID IMPLEMENTATION

HS_MAIN.H

```
/* hs_main.h */

int main();
double calculateSystemEnergy();
double interactionEnergy(int i, int j);
void perturbSystem();
void updatePosition();
int checkForOverlap(int particle_number);
```

HS_MAIN.C

```
/************************************* hs_main.c
*****************************************/

#include "hs_main.h"
#include "io_setup.h"
#include "graphics.h"

#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <ftw_science.h>
#include <ftw_std.h>
#include <ftw_rng.h>
#include <ftw_param.h>

#ifndef MAX_NUMBER_MOLECULES
#define MAX_NUMBER_MOLECULES 16384
#endif

/* non-configurable global params */
double x[MAX_NUMBER_MOLECULES], y[MAX_NUMBER_MOLECULES],
z[MAX_NUMBER_MOLECULES];
int wsize_x, wsize_y, wsize_z;
int change_flag = TRUE; /* signals need to update display */
int monte_carlo_steps = 0;
int monte_carlo_step_counter = 0;

double acceptance_ratio;
double perturbation_length = .2;
int attempted_moves;
int accepted_moves;
int number_of_pairs;
int verbose = 0;
int graphics = 1;
int side_view = 1;
int particle_scale = 64; /* how many pixels */
int number_of_molecules = 25;
double box_x = 7.0;
```

```
double box_y = 7.0;
double box_z = 7.0;
int fg_color = 255;
int bg_color = 0;
int min_color = 64;
int rng_seed = 24375;
/* L-J params are chosen such that intercepts are at 1, 2.5 */
int energy_report_frequency = 100;
int running_average_steps = 1000;
double running_average = 0.0;
int end_mcs = 50000; /* exit simulation after reaching this pt */
int configuration_threshold = 0; /* start to dump config after this many mcs
*/
int configuration_frequency = 1000;
char *output_file_name = "hs.out";
char *input_file_name;
char *log_file_name = "hs.log";
char *simulation_unique_identifier = "################";
char hostname[50] = "";
double target_acceptance_ratio = .15;
extern double initial_spacing;

char *display_name_1 = "X-Y projection (front)";
char *display_name_2 = "Z-Y projection (right side)";

int mirror_depth = 1;

int main(int argc, char *argv[])
{
  setCommandLineParameters(argc, argv);
  verbose = getFlagParam("-v");
  if(getFlagParam("-ng")) graphics = 0;
  if(getFlagParam("-no_side")) {side_view = 0; graphics = 1;}
  getIntParam("-particle_scale", &particle_scale);
  getIntParam("-N", &number_of_molecules);
  getVectorParam("-box", &box_x, &box_y, &box_z);
  getIntParam("-fg_color", &fg_color);
  getIntParam("-bg_color", &bg_color);
  getIntParam("-min_color", &min_color);
  getIntParam("-rng_seed", &rng_seed);
  if (getFlagParam("-randomize")) rng_seed = getRandomSeed();
  getIntParam("-end_mcs", &end_mcs);
  getIntParam("-energy_report_frequency", &energy_report_frequency);
  getIntParam("-configuration_threshold", &configuration_threshold);
  getIntParam("-configuration_frequency", &configuration_frequency);
  getStringParam("-log_file_name", &log_file_name);
  getStringParam("-input_file_name", &input_file_name);
  getStringParam("-simulation_unique_identifier",
&simulation_unique_identifier);
  getDoubleParam("-target_acceptance_ratio", &target_acceptance_ratio);
  getDoubleParam("-initial_spacing", &initial_spacing);

  initializeRandomNumberGeneratorTo(rng_seed);

  readEnvironmentVariables();
  initializeOutput();
  setInitialConditions();
```

```
  if (graphicsModeEnabled()) initializeDisplay();

  for(monte_carlo_steps=0; monte_carlo_steps<=end_mcs; monte_carlo_steps++)
  {
    generateOutput();
    attempted_moves = 0;
    accepted_moves = 0;
    for (monte_carlo_step_counter=0;
monte_carlo_step_counter<number_of_molecules; monte_carlo_step_counter++)
perturbSystem();
    acceptance_ratio = (0.0 + accepted_moves)/(0.0 + attempted_moves);
    if (graphicsModeEnabled() && changeFlagIsSet())
drawGraphicalRepresentation();
  }

  finalizeOutput();
  return 0;
} /* end main */

/* generate a move, check for overlap */
void perturbSystem()
{
  double dx, dy, dz;
  int particle_number;

  /* which molecule and how much to move it */
  attempted_moves++;
  particle_number = floor(number_of_molecules * rnd());
  dx = (rnd() -.5) * perturbation_length;
  dy = (rnd() -.5) * perturbation_length;
  dz = (rnd() -.5) * perturbation_length;

  updatePosition(particle_number, dx, dy, dz);

  if (checkForOverlap(particle_number))
  {
    updatePosition(particle_number, -dx, -dy, -dz);
  }
  else
  {
    change_flag = 1;
    accepted_moves++;
  }
}

void updatePosition(int particle_number, double dx, double dy, double dz)
{
//printf("moving #%d by %lf, %lf, %lf\n", particle_number, dx, dy, dz);
//sleep(1);
  x[particle_number] += dx;
  if (x[particle_number] > box_x) x[particle_number] -= box_x;
  if (x[particle_number] < 0) x[particle_number] = x[particle_number] +
box_x;
  y[particle_number] += dy;
  if (y[particle_number] > box_y) y[particle_number] -= box_y;
  if (y[particle_number] < 0) y[particle_number] += box_y;
  z[particle_number] += dz;
```

```
  if (z[particle_number] > box_z) z[particle_number] -= box_z;
  if (z[particle_number] < 0) z[particle_number] += box_z;
}

int checkForOverlap(int particle_number)
{
  int i;
  int mirror_x, mirror_y, mirror_z;
  double dist_x, dist_y, dist_z;
  double dist_sq;

  for (i=0; i<number_of_molecules; i++)
  {
    if (i==particle_number) continue;
    for (mirror_x=-mirror_depth; mirror_x<=mirror_depth; mirror_x++)
    for (mirror_y=-mirror_depth; mirror_y<=mirror_depth; mirror_y++)
    for (mirror_z=-mirror_depth; mirror_z<=mirror_depth; mirror_z++)
    {
      dist_x = mirror_x*box_x + x[particle_number] - x[i];
      dist_y = mirror_y*box_y + y[particle_number] - y[i];
      dist_z = mirror_z*box_z + z[particle_number] - z[i];

      dist_sq = dist_x*dist_x + dist_y*dist_y + dist_z*dist_z;
      if (dist_sq < 1) return 1;
    }
  }

  return 0;
}
```

COMMAND_LINE_PARSER.H

```
void parseCommandLineOptions(int argc, char *argv[]);
void printUsage();
```

COMMAND_LINE_PARSER.C

```
#include <stdio.h>
#include <math.h>
#include <string.h>
#include <stdlib.h>
#include <ftw_std.h>
#include <ftw_rng.h>

#include "command_line_parser.h"

/* parameters configurable on command line and default values */
int verbose = 0;
int graphics = 1;
int side_view = 1;
double temperature = 1;
int particle_scale = 64; /* how many pixels */
int number_of_molecules = 25;
double box_x = 7.0;
double box_y = 7.0;
```

```
double box_z = 7.0;
int fg_color = 255;
int bg_color = 0;
int min_color = 64;
int rng_seed = 24375;
/* L-J params are chosen such that intercepts are at 1, 2.5 */
int energy_report_frequency = 100;
int running_average_steps = 1000;
double running_average = 0.0;
int end_mcs = 50000; /* exit simulation after reaching this pt */
int configuration_threshold = 0; /* start to dump config after this many mcs
*/
int configuration_frequency = 1000;
char *output_file_name = "hs.out";
char *input_file_name;
char *log_file_name = "hs.log";
char simulation_unique_identifier[25] = "################";
char hostname[50] = "";
double target_acceptance_ratio = .15;
extern double initial_spacing;

char *display_name_1 = "X-Y projection (front)";
char *display_name_2 = "Z-Y projection (right side)";

void parseCommandLineOptions(int argc, char *argv[])
{
  int i;

  for (i = 0; i<argc; i++)
  {
    if (!strcmp(argv[i], "-usage")) printUsage();
    if (!strcmp(argv[i], "-v")) verbose = 1;
    if (!strcmp(argv[i], "-ng")) graphics = 0;
    if (!strcmp(argv[i], "-no_side")) {side_view = 0; graphics = 1;}
    if (!strcmp(argv[i], "-T")) temperature = strtod(argv[++i], NULL);
    if (!strcmp(argv[i], "-initial_spacing")) initial_spacing =
strtod(argv[++i], NULL);
    if (!strcmp(argv[i], "-particle_scale")) particle_scale =
strtol(argv[++i], NULL, 10);
    if (!strcmp(argv[i], "-N")) number_of_molecules = strtol(argv[++i], NULL,
10);
    if (!strcmp(argv[i], "-box"))
    {
       box_x = strtod(argv[++i], NULL);
       box_y = strtod(argv[++i], NULL);
       box_z = strtod(argv[++i], NULL);
    }
    if (!strcmp(argv[i], "-fg_color")) fg_color = strtol(argv[++i], NULL,
10);
    if (!strcmp(argv[i], "-bg_color")) bg_color = strtol(argv[++i], NULL,
10);
    if (!strcmp(argv[i], "-min_color")) min_color = strtol(argv[++i], NULL,
10);
    if (!strcmp(argv[i], "-rng_seed")) rng_seed = strtol(argv[++i], NULL,
10);
    if (!strcmp(argv[i], "-randomize")) rng_seed = getRandomSeed();
    if (!strcmp(argv[i], "-end_mcs")) end_mcs = strtod(argv[++i], NULL);
```

```
    if (!strcmp(argv[i], "-energy_report_frequency")) energy_report_frequency
= strtol(argv[++i], NULL, 10);
    if (!strcmp(argv[i], "-configuration_threshold")) configuration_threshold
= strtol(argv[++i], NULL, 10);
    if (!strcmp(argv[i], "-configuration_frequency")) configuration_frequency
= strtol(argv[++i], NULL, 10);
    if (!strcmp(argv[i], "-log_file_name")) log_file_name = argv[++i];
    if (!strcmp(argv[i], "-input_file_name")) input_file_name = argv[++i];
    if (!strcmp(argv[i], "-simulation_unique_identifier"))
strcpy(simulation_unique_identifier, argv[++i]);
    if (!strcmp(argv[i], "-target_acceptance_ratio")) target_acceptance_ratio
= strtod(argv[++i], NULL);
  }

  initializeRandomNumberGeneratorTo(rng_seed);

//  if (!strcmp(simulation_unique_identifier, "################"))
//    for (i=0; i<16; i++) *(simulation_unique_identifier + i) = (char)(rnd()
* 26 + 65);
}

void printUsage()
{
  printf("usage:  hs [-options]\n");
  exit(0);
}
```

IO_SETUP.H

```
/* io_setup.h */

void setInitialConditions();
void generateUniqueId();
void finalizeOutput();
void initializeOutput();
void generateOutput();
void loadConfiguration();
void readEnvironmentVariables();
```

IO_SETUP.C

```
/* io_setup.c */

#include <ftw_std.h>
#include <ftw_rng.h>
#include <time.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

#include "io_setup.h"
#include "hs_main.h"

extern char simulation_unique_identifier[];
```

```
extern double temperature;
extern int number_of_molecules;
extern int energy_report_frequency;
extern int configuration_threshold;
extern int configuration_frequency;
extern double box_x, box_y, box_z;
extern int monte_carlo_steps;
extern int end_mcs;
extern char* log_file_name;
extern char* output_file_name;
extern char* input_file_name;
extern double x[], y[], z[];

double initial_spacing = 1.03;

FILE *log_file;
FILE *output_file;
FILE *input_file;

char hostname[50];
char *log_path;
char *results_path;

time_t now;

void setFCCInitialCondition()
{
  double root2 = 1.414213562373;
  double xx, yy, zz;
  int n=0;
  double lattice_param = 2*initial_spacing/root2;

  for (xx=0; xx<box_x-lattice_param; xx += lattice_param)
  for (yy=0; yy<box_y-lattice_param; yy += lattice_param)
  for (zz=0; zz<box_z-lattice_param; zz += lattice_param)
  {
    if (n<number_of_molecules)
    {
      x[n] = xx;
      y[n] = yy;
      z[n] = zz;
      n++;
    }
    if (n<number_of_molecules)
    {
      x[n] = xx;
      y[n] = yy + lattice_param/2;
      z[n] = zz + lattice_param/2;
      n++;
    }
    if (n<number_of_molecules)
    {
      x[n] = xx + lattice_param/2;
      y[n] = yy;
      z[n] = zz + lattice_param/2;
      n++;
    }
```

```
    if (n<number_of_molecules)
    {
      x[n] = xx + lattice_param/2;
      y[n] = yy + lattice_param/2;
      z[n] = zz;
      n++;
    }
  }

  if (n<number_of_molecules)
  {
    printf("too many molecules... only %d/%d placed.\n", n,
number_of_molecules);
    exit(1);
  }
}

void XXsetInitialConditions()
{
  int target_number_of_molecules;

  if (input_file_name != NULL) loadConfiguration();
  else
  {
    monte_carlo_steps = 0;
    target_number_of_molecules = number_of_molecules;
    for (number_of_molecules=0;
number_of_molecules<target_number_of_molecules; number_of_molecules++)
    {
      while (1)
      {
        x[number_of_molecules] = rnd() * box_x;
        y[number_of_molecules] = rnd() * box_y;
        z[number_of_molecules] = rnd() * box_z;

        if (!checkForOverlap(number_of_molecules)) break;
      }
printf("added #%d\n", number_of_molecules);
    }
  }
}

void setInitialConditions()
{
  //int xx, yy, zz;
  //int num_so_far=0;

  if (input_file_name != NULL) loadConfiguration();
  else setFCCInitialCondition();
/*
  {
    monte_carlo_steps = 0;
    for (xx=0; xx<box_x; xx+=1.1)
    for (j=0; j<box_y; j++)
    for (k=0; k<box_z; k++)
    {
      x[num_so_far] = i*1.1;
```

```
      y[num_so_far] = j*1.1;
      z[num_so_far] = k*1.1;

      if (num_so_far++ >= number_of_molecules) return;
    }

    printf ("too many molecules...\n");
    exit(0);
  }
*/
}

void loadConfiguration()
{
  FILE *datastream;
  char line[80];
  char *xs, *ys, *zs;

  number_of_molecules = 0;
  V printf("loading %s...\n", input_file_name);
  datastream = fopen(input_file_name, "r");

  while (1)
  {
    fgets(line, 80, datastream);
    if (feof(datastream)) break;

    xs = strtok(line, "\t");
    ys = strtok(NULL, "\t");
    zs = strtok(NULL, "\n");

    x[number_of_molecules] = strtod(xs, NULL);
    y[number_of_molecules] = strtod(ys, NULL);
    z[number_of_molecules++] = strtod(zs, NULL);
  }

  V printf("%d lines read.\n", number_of_molecules);
  fclose(datastream);
}

void generateUniqueId()
{
  int i;
  if (!strcmp(simulation_unique_identifier, "################"))
    for (i=0; i<16; i++) *(simulation_unique_identifier + i) = (char)(rnd() *
26 + 65);
}

void readEnvironmentVariables()
{
  gethostname(hostname, 50);
  log_path = getenv("LOG_PATH");
  log_file_name = strcat(log_path, "/");
  log_file_name = strcat(log_file_name, hostname);
  log_file_name = strcat(log_file_name, "-hs.log");
  results_path = getenv("RESULTS_PATH");
  output_file_name = strcat(results_path, "/");
```

```
  output_file_name = strcat(output_file_name, simulation_unique_identifier);
  output_file_name = strcat(output_file_name, "-hs.out");
}

/* display headers on output */
void initializeOutput()
{
  now = time(NULL);

  if (verbose)
  {
    printf("#HT simulation %s started on %s:  %s",
simulation_unique_identifier, hostname, ctime(&now));

printf("#HHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHH\n")
;
    printf("#H N=%d\n", number_of_molecules);
    printf("#H box dimensions:  \t%lf x %lf x %lf = %lf\n", box_x, box_y,
box_z, box_x*box_y*box_z);
    printf("#H reduced density = %lf\n", (number_of_molecules / (box_x *
box_y * box_z)));
    printf("#H\n");
    printf("#H configuration threshold:\t%d\n", configuration_threshold);
    printf("#H configuration frequency:\t%d\n", configuration_frequency);
    printf("#H run until mcs = %d\n", end_mcs);

printf("#HHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHH\n")
;
    printf("#H\n");
    printf("#H\n");
  }

  log_file = fopen(log_file_name, "a");
  fprintf(log_file, "simulation %s launched on %s:  %s",
simulation_unique_identifier, hostname, ctime(&now));
  fclose(log_file);

  output_file = fopen(output_file_name, "w");
  fprintf(output_file, "#HT simulation %s started on %s:  %s",
simulation_unique_identifier, hostname, ctime(&now));
  fprintf(output_file,
"#HHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHH\n");
  fprintf(output_file, "#H N=%d\n", number_of_molecules);
  fprintf(output_file, "#H box dimensions:  \t%lf x %lf x %lf = %lf\n",
box_x, box_y, box_z, box_x*box_y*box_z);
  fprintf(output_file, "#H reduced density = %lf\n", (number_of_molecules /
(box_x * box_y * box_z)));
  fprintf(output_file, "#H\n");
  fprintf(output_file, "#H configuration threshold:\t%d\n",
configuration_threshold);
  fprintf(output_file, "#H configuration frequency:\t%d\n",
configuration_frequency);
  fprintf(output_file, "#H run until mcs = %d\n", end_mcs);
  fprintf(output_file,
"#HHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHH\n");
  fprintf(output_file, "#H\n");
  fprintf(output_file, "#H\n");
```

```
  fflush(output_file);
}

void finalizeOutput()
{
  now = time(NULL);
  if (verbose) printf("#HT simulation %s finished on %s:  %s",
simulation_unique_identifier, hostname, ctime(&now));
  fprintf(output_file, "#HT simulation %s finished on %s:  %s",
simulation_unique_identifier, hostname, ctime(&now));
  fclose(output_file);

  log_file = fopen(log_file_name, "a");
  fprintf(log_file, "simulation %s finished on %s:  %s",
simulation_unique_identifier, hostname, ctime(&now));
  fclose(log_file);
}

void generateOutput()
{
  int i;

  if ((monte_carlo_steps > configuration_threshold) && (monte_carlo_steps %
configuration_frequency == 0))
  {
    now = time(NULL);
    log_file = fopen(log_file_name, "a");
    fprintf(log_file, "dumping configuration for %s on %s at %d steps:  %s",
hostname, \
            simulation_unique_identifier, monte_carlo_steps, ctime(&now));
    fclose(log_file);

    if (verbose)
    {
      printf("#HC%06d\n", monte_carlo_steps);
      printf("#HC%06d dumping configuration at mcs=%d...\n",
monte_carlo_steps, monte_carlo_steps);
      printf("#HC%06d\n", monte_carlo_steps);
      for (i=0; i<number_of_molecules; i++)
printf("#C%06d\t%d\t%lf\t%lf\t%lf\n", monte_carlo_steps, i, x[i], y[i],
z[i]);
      printf("#HC%06d\n", monte_carlo_steps);
    }

    fprintf(output_file, "#HC%06d\n", monte_carlo_steps);
    fprintf(output_file, "#HC%06d dumping configuration at mcs=%d...\n",
monte_carlo_steps, monte_carlo_steps);
    fprintf(output_file, "#HC%06d\n", monte_carlo_steps);
    for (i=0; i<number_of_molecules; i++) fprintf(output_file,
"#C%06d\t%d\t%lf\t%lf\t%lf\n", monte_carlo_steps, i, x[i], y[i], z[i]);
    fprintf(output_file, "#HC%06d\n", monte_carlo_steps);
  }

  fflush(output_file);
}
```

GRAPHICS.H

```
/* graphics.h */

#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include <X11/Xos.h>
#include <X11/keysym.h>
#include <X11/Xresource.h>
#include <stdlib.h>
#include <stdio.h>
#include <math.h>

void checkForWindowEvent();
void initializeDisplay();
void setChangeFlag();
void resetChangeFlag();
int changeFlagIsSet();
int graphicsModeEnabled();
void drawGraphicalRepresentation();
```

GRAPHICS.C

```
/* graphics.c */

#include "graphics.h"
#include <ftw_science.h>
#include <ftw_std.h>

extern int side_view;
extern int wsize_x, wsize_y, wsize_z;
extern int particle_scale;
extern double box_x, box_y, box_z;
extern int change_flag;
extern int graphics;
extern int number_of_molecules;
extern int bg_color;
extern int fg_color;
extern int min_color;
extern double x[], y[], z[];

Display *dpy;
Window window, window2;
GC context, context2;
XEvent event;
XGCValues gcvalues;
char *displayname="simulation";
char *display=NULL;
int *z_rank;
int *x_rank;

extern char *display_name_1;
extern char *display_name_2;

int change_flag;
```

```
void checkForWindowEvent()
{
  int rstat;

  rstat = XCheckMaskEvent(dpy, ExposureMask | ButtonPressMask, &event);
  if (rstat&&(event.type == ButtonPress))
  {
    XFreeGC(dpy, context);
    XCloseDisplay(dpy);
    exit(0);
  }
}

void initializeDisplay()
{
  wsize_x = (int)(particle_scale * box_x);
  wsize_y = (int)(particle_scale * box_y);
  wsize_z = (int)(particle_scale * box_z);

  z_rank = (int*)malloc(sizeof(int) * number_of_molecules);
  x_rank = (int*)malloc(sizeof(int) * number_of_molecules);

  dpy = XOpenDisplay(display);
  if (dpy == NULL) printf("Can't open the display.\n");

  window = XCreateSimpleWindow(dpy, DefaultRootWindow(dpy), 0, 0, wsize_x,
wsize_y, 0,0,0);
  XSelectInput(dpy, window,
(StructureNotifyMask|ExposureMask|ButtonPressMask|ButtonReleaseMask));
  XMapWindow(dpy, window);
  context = XCreateGC(dpy, window, GCForeground | GCBackground, &gcvalues);
  XStoreName(dpy, window, display_name_1);
  XSetIconName(dpy, window, display_name_2);

  if (side_view)
  {
    window2 = XCreateSimpleWindow(dpy, DefaultRootWindow(dpy), wsize_x, 0,
wsize_z, wsize_y, 0,0,0);
    XSelectInput(dpy, window2,
(StructureNotifyMask|ExposureMask|ButtonPressMask|ButtonReleaseMask));
    XMapWindow(dpy, window2);
    context2 = XCreateGC(dpy, window2, GCForeground | GCBackground,
&gcvalues);
    XStoreName(dpy, window2, display_name_2);
    XSetIconName(dpy, window2, display_name_2);
  }

  checkForWindowEvent();
}

void setChangeFlag()
{
  change_flag = TRUE;
}

void resetChangeFlag()
{
```

```
  change_flag = FALSE;
}

int changeFlagIsSet()
{
  return change_flag;
}

int graphicsModeEnabled()
{
  return graphics;
}

void drawGraphicalRepresentation()
{
  int xx, yy, zz;
  int i, j;
  int z_shade, x_shade;
  int temporary;

  if (!changeFlagIsSet()) return;

  for (i=0; i<number_of_molecules; i++) z_rank[i] = i;
  gcvalues.foreground = bg_color;
  XChangeGC(dpy, context, GCForeground, &gcvalues);
  XFillRectangle(dpy, window, context, 0, 0, wsize_x*2, wsize_y*2);

  /* order by z values */
  for (i=number_of_molecules; i>0; i--)
    for (j=0; j<i-1; j++)
      if (z[z_rank[j]] < z[z_rank[j+1]])
      {
        temporary = z_rank[j];
        z_rank[j] = z_rank[j+1];
        z_rank[j+1] = temporary;
      }

  for(i=0;i<number_of_molecules;i++)
  {
    xx=floor(x[z_rank[i]]*particle_scale - particle_scale/2);
    yy=floor(y[z_rank[i]]*particle_scale - particle_scale/2);
    z_shade = fg_color - floor((fg_color - min_color) * z[z_rank[i]] /
box_z);
    gcvalues.foreground = z_shade;
    XChangeGC(dpy, context, GCForeground, &gcvalues);
    XFillArc(dpy, window, context, xx, yy, particle_scale, particle_scale, 0,
360*64);
    XFillArc(dpy, window, context, xx, yy + wsize_y, particle_scale,
particle_scale, 0, 360*64);
    XFillArc(dpy, window, context, xx + wsize_x, yy, particle_scale,
particle_scale, 0, 360*64);
    XFillArc(dpy, window, context, xx + wsize_x, yy + wsize_y,
particle_scale, particle_scale, 0, 360*64);
  }

  if (side_view)
  {
```

```
    for (i=0; i<number_of_molecules; i++) x_rank[i] = i;
    gcvalues.foreground = bg_color;
    XChangeGC(dpy, context2, GCForeground, &gcvalues);
    XFillRectangle(dpy, window2, context2, -wsize_z/2, -wsize_y/2, wsize_z*2,
wsize_y*2);

    /* order by x values */
    for (i=number_of_molecules; i>0; i--)
      for (j=0; j<i-1; j++)
        if (x[x_rank[j]] > x[x_rank[j+1]])
        {
          temporary = x_rank[j];
          x_rank[j] = x_rank[j+1];
          x_rank[j+1] = temporary;
        }

    for(i=0; i<number_of_molecules;i++)
    {
      x_shade = min_color + floor((fg_color - min_color) * x[x_rank[i]] /
box_x);
      yy=floor(y[x_rank[i]]*particle_scale - particle_scale/2);
      zz=floor(z[x_rank[i]]*particle_scale - particle_scale/2);
      gcvalues.foreground = x_shade;
      XChangeGC(dpy, context2, GCForeground, &gcvalues);
      XFillArc(dpy, window2, context2, zz, yy, particle_scale,
particle_scale, 0, 360*64);
      XFillArc(dpy, window2, context2, zz, yy + wsize_y, particle_scale,
particle_scale, 0, 360*64);
      XFillArc(dpy, window2, context2, wsize_z + zz, yy, particle_scale,
particle_scale, 0, 360*64);
      XFillArc(dpy, window2, context2, wsize_z + zz, yy + wsize_y,
particle_scale, particle_scale, 0, 360*64);
    }
  }

  XFlush(dpy);
  checkForWindowEvent();
  resetChangeFlag();
}
```

REFERENCES

1. Willmore, F. T., Wang, X.Y., Sanchez, I.C., Diffusion in glassy polymers via probabilistic molecular dynamics. *submitted to Physical Review Letters* **2006**.
2. Willmore, F. T.; Wang, X. Y.; Sanchez, I. C., Free volume properties of model fluids and polymers: Shape and connectivity. *Journal of Polymer Science Part B-Polymer Physics* **2006,** 44, (9), 1385-1393.
3. Wang, X. Y.; Willmore, F. T.; Raharjo, R. D.; Wang, X. C.; Freeman, B. D.; Hill, A. J.; Sanchez, I. C., Molecular simulations of physical aging in polymer membrane materials. *Journal of Physical Chemistry B* **2006,** 110, (33), 16685-16693.
4. van Krevelen, W. D., *Properties of Polymers*. 3rd edition ed.; Elsevier: Amsterdam, 1997.
5. Cohen, M. H.; Turnbull, D., Molecular Transport in liquids and gases. *J Phys. Chem.* **1959,** 31, 1164-1169.
6. Thran, A.; Kroll, G.; Faupel, F., Correlation between fractional free volume and diffusivity of gas molecules in classy polymers. *Journal of Polymer Science Part B-Polymer Physics* **1999,** 37, (23), 3344-3358.
7. In 't Veld, P. J.; Stone, M. T.; Truskett, T. M.; Sanchez, I. C., Liquid structure via cavity size distributions. *Journal of Physical Chemistry B* **2000,** 104, (50), 12028-12034.
8. Wang, X. Y.; Lee, K. M.; Lu, Y.; Stone, M. T.; Sanchez, I. C.; Freeman, B. D., Cavity size distributions in high free volume glassy polymers by molecular simulation. *Polymer* **2004,** 45, (11), 3907-3912.
9. Wang, X. Y.; Raharjo, R. D.; Lee, H. J.; Lu, Y.; Freeman, B. D.; Sanchez, I. C., Molecular simulation and experimental study of substituted polyacetylenes: Fractional free volume, cavity size distributions and diffusion coefficients. *Journal of Physical Chemistry B* **2006,** 110, (25), 12666-12672.
10. Widom, B., Some Topics in the Theory of Fluids. *The Journal of Chemical Physics* **1963,** 39, (11), 2808-2812.
11. Hill, A. J.; Freeman, B. D.; Jaffe, M.; Merkel, T. C.; Pinnau, I., Tailoring nanospace. *Journal of Molecular Structure* **2005,** 739, (1-3), 173-178.
12. Hill, A. J.; Meakin, P.; Freeman, B. D., Modeling the relationship between free volume and transport in polymers: Theory and experiment. *Abstracts of Papers of the American Chemical Society* **2002,** 223, D74-D74.
13. Hill, A. J.; Nagai, K.; Freeman, B. D., Free volume dynamics and physical aging of high-permeability membrane polymers. *Abstracts of Papers of the American Chemical Society* **1999,** 218, U709-U709.
14. Sega, M.; Jedlovszky, P.; Medvedev, N. N.; Vallauri, R., Free volume properties of a linear soft polymer: A computer simulation study. *Journal of Chemical Physics* **2004,** 121, (5), 2422-2427.
15. Shantarovich, V. P.; Azamatova, Z. K.; Novikov, Y. A., Positron investigations of free-volume elements in polymer gas-separation membranes. *Physics of the Solid State* **1998,** 40, (1), 147-149.
16. Shantarovich, V. P.; Azamatova, Z. K.; Novikov, Y. A.; Yampolskii, Y. P., Free-volume distribution of high permeability membrane materials probed by positron annihilation. *Macromolecules* **1998,** 31, (12), 3963-3966.

17. Shantarovich, V. P.; Kleiner, V. I.; Alent'ev, A. Y.; Kevdina, I. B.; Azamatova, Z. K.; Filimonov, M. K., Positronium formation and structural irregularities of polymeric materials. *High Energy Chemistry* **1998,** 32, (1), 50-54.
18. Shantarovich, V. P.; Yampolskii, Y. P.; Kevdina, I. B.; Azamatova, Z. K.; Khotimskii, V. S., Examination of the free volume elements in polymeric systems by positron annihilation spectroscopy. *Vysokomolekulyarnye Soedineniya Seriya a & Seriya B* **1997,** 39, (3), 445-450.
19. Wang, C. L.; Hirade, T.; Maurer, F. H. J.; Eldrup, M.; Pedersen, N. J., Free-volume distribution and positronium formation in amorphous polymers: Temperature and positron-irradiation-time dependence. *Journal of Chemical Physics* **1998,** 108, (11), 4654-4661.
20. Jordan, S. S.; Koros, W. J., A Free-Volume Distribution Model of Gas Sorption and Dilation in Glassy-Polymers. *Macromolecules* **1995,** 28, (7), 2228-2235.
21. Golemme, G.; Nagy, J. B.; Fonseca, A.; Algieri, C.; Yampolskii, Y., Xe-129-NMR study of free volume in amorphous perfluorinated polymers: comparsion with other methods. *Polymer* **2003,** 44, (17), 5039-5045.
22. Corti, D. S.; Debenedetti, P. G.; Sastry, S.; Stillinger, F. H., Constraints, metastability, and inherent structures in liquids. *Physical Review E* **1997,** 55, (5), 5522-5534.
23. Lee, S.; Mattice, W. L., A "phantom bubble" model for the distribution of free volume in polymers. *Computational and Theoretical Polymer Science* **1999,** 9, (1), 57-61.
24. Sastry, S.; Corti, D. S.; Debenedetti, P. G.; Stillinger, F. H., Statistical geometry of particle packings .1. Algorithm for exact determination of connectivity, volume, and surface areas of void space in monodisperse and polydisperse sphere packings. *Physical Review E* **1997,** 56, (5), 5524-5532.
25. Sastry, S.; Debenedetti, P. G.; Stillinger, F. H.; Schroder, T. B.; Dyre, J. C.; Glotzer, S. C., Potential energy landscape signatures of slow dynamics in glass forming liquids. *Physica a-Statistical Mechanics and Its Applications* **1999,** 270, (1-2), 301-308.
26. Sastry, S.; Truskett, T. M.; Debenedetti, P. G.; Torquato, S.; Stillinger, F. H., Free volume in the hard sphere liquid. *Molecular Physics* **1998,** 95, (2), 289-297.
27. Starr, F. W.; Sastry, S.; Douglas, J. F.; Glotzer, S. C., What do we learn from the local geometry of glass-forming liquids? *Physical Review Letters* **2002,** 89, (12).
28. Curco, D.; Zanuy, D.; Aleman, C., EVEBAT: A fast strategy for the examination of the empty space in polymer matrices. *Journal of Computational Chemistry* **2003,** 24, (10), 1208-1214.
29. Gusev, A. A.; Arizzi, S.; Suter, U. W.; Moll, D. J., Dynamics of Light Gases in Rigid Matrices of Dense Polymers. *Journal of Chemical Physics* **1993,** 99, (3), 2221-2227.
30. Gusev, A. A.; Suter, U. W., Dynamics of Small Molecules in Dense Polymers Subject to Thermal Motion. *Journal of Chemical Physics* **1993,** 99, (3), 2228-2234.
31. Gusev, A. A.; Suter, U. W., Modeling of Solid Polymers in Chemical Detail - Gases in Amorphous Polymers. *Makromolekulare Chemie-Macromolecular Symposia* **1993,** 69, 229-236.
32. Gusev, A. A.; Suter, U. W.; Moll, D. J., Relationship between Helium Transport and Molecular Motions in a Glassy Polycarbonate. *Macromolecules* **1995,** 28, (7), 2582-2584.
33. Lim, S. Y.; Tsotsis, T. T.; Sahimi, M., Molecular simulation of diffusion and sorption of gases in an amorphous polymer. *Journal of Chemical Physics* **2003,** 119, (1), 496-504.
34. Wang, X.-Y.; in 't Veld, P. J.; Lu, Y.; Freeman, B. D.; Sanchez, I. C., A molecular simulation study of cavity size distributions and diffusion in para and meta isomers. *Polymer* **2005,** 46, (21), 9155-9161.

35. Middleton, T. F.; Wales, D. J., Comparison of kinetic Monte Carlo and molecular dynamics simulations of diffusion in a model glass former. *Journal of Chemical Physics* **2004,** 120, (17), 8134-8143.
36. Heuchel, M.; Hofmann, D., Molecular modelling of polyimide membranes for gas separation. *Desalination* **2002,** 144, (1-3), 67-72.
37. Heuchel, M.; Hofmann, D.; Pullumbi, P., Molecular Modeling of small-molecule permeation in polyimides and its correlation to free-volume distributions. *Macromolecules* **2004,** 37, (1), 201-214.
38. Hofmann, D.; Entrialgo-Castano, M.; Lerbret, A.; Heuchel, M.; Yampolskii, Y., Molecular modeling investigation of free volume distributions in stiff chain polymers with conventional and ultrahigh free volume: Comparison between molecular modeling and positron lifetime studies. *Macromolecules* **2003,** 36, (22), 8528-8538.
39. Hofmann, D.; Heuchel, M.; Yampolskii, Y.; Khotimskii, V.; Shantarovich, V., Free volume distributions in ultrahigh and lower free volume polymers: Comparison between molecular modeling and positron lifetime studies. *Macromolecules* **2002,** 35, (6), 2129-2140.
40. Tamai, Y.; Fukuda, M., Effect of encaged aromatic guests on the shape and connectivity of molecular cavity in crystalline polystyrene evaluated by molecular simulations. *Journal of Chemical Physics* **2004,** 121, (23), 12085-12093.
41. Budd, P. M.; McKeown, N. B.; Fritsch, D., Free volume and intrinsic microporosity in polymers. *Journal of Materials Chemistry* **2005,** 15, (20), 1977-1986.
42. Sok, R. M.; Berendsen, H. J. C.; Vangunsteren, W. F., Molecular-Dynamics Simulation of the Transport of Small Molecules across a Polymer Membrane. *Journal of Chemical Physics* **1992,** 96, (6), 4699-4704.
43. Tsige, M.; Grest, G. S., Molecular dynamics simulation of solvent-polymer interdiffusion: Fickian diffusion. *Journal of Chemical Physics* **2004,** 120, (6), 2989-2995.
44. Tsige, M.; Grest, G. S., Interdiffusion of solvent into glassy polymer films: A molecular dynamics study. *Journal of Chemical Physics* **2004,** 121, (15), 7513-7519.
45. Rico-Martinez, R.; Gear, C. W.; Kevrekidis, I. G., Coarse projective kMC integration: forward/reverse initial and boundary value problems. *Journal of Computational Physics* **2004,** 196, (2), 474-489.
46. Vigil, R. D.; Willmore, F. T., Oscillatory dynamics in a heterogeneous surface reaction: Breakdown of the mean-field approximation. *Physical Review E* **1996,** 54, (2), 1225-1231.
47. Gauthier, M. G.; Slater, G. W., Building reliable lattice Monte Carlo models for real drift and diffusion problems. *Physical Review E* **2004,** 70, (1).
48. Gauthier, M. G.; Slater, G. W., A new set of Monte Carlo moves for lattice random-walk models of biased diffusion. *Physica a-Statistical Mechanics and Its Applications* **2005,** 355, (2-4), 283-296.
49. Gauthier, M. G.; Slater, G. W.; Dorfman, K. D., Exact lattice calculations of dispersion coefficients in the presence of external fields and obstacles. *European Physical Journal E* **2004,** 15, (1), 71-82.
50. Armatas, G. S.; Petrakis, D. E.; Pomonis, P. J., Estimation of diffusion parameters in functionalized silicas with modulated porosity - Part II: Pore network modeling. *Journal of Chromatography A* **2005,** 1074, (1-2), 61-69.
51. Zielinski, J. M.; Duda, J. L., Predicting Polymer Solvent Diffusion-Coefficients Using Free-Volume Theory. *Aiche Journal* **1992,** 38, (3), 405-415.

52. Nagel, C.; Schmidtke, E.; Gunther-Schade, K.; Hofmann, D.; Fritsch, D.; Strunskus, T.; Faupel, F., Free volume distributions in glassy polymer membranes: Comparison between molecular modeling and experiments. *Macromolecules* **2000,** 33, (6), 2242-2248.
53. Bharadwaj, R. K.; Boyd, R. H., Small molecule penetrant diffusion in aromatic polyesters: a molecular dynamics simulation study. *Polymer* **1999,** 40, (15), 4229-4236.
54. Takeuchi, H.; Okazaki, K., Molecular-Dynamics Simulation of Diffusion of Simple Gas Molecules in a Short Chain Polymer. *Journal of Chemical Physics* **1990,** 92, (9), 5643-5652.
55. Han, J.; Boyd, R. H., Molecular packing and small-penetrant diffusion in polystyrene: A molecular dynamics simulation study. *Polymer* **1996,** 37, (10), 1797-1804.
56. Wikipedia, **N/A**.
57. Jarzynski, C., Equilibrium free energies from nonequilibrium processes. *Acta Physica Polonica B* **1998,** 29, (6), 1609-1622.
58. Jarzynski, C., Equilibrium free-energy differences from nonequilibrium measurements: A master-equation approach. *Physical Review E* **1997,** 56, (5), 5018-5035.
59. Metropolis, N., Equation of State Calculations by Fast Computing Machines. *J Chem Phys* **1953,** 21, 1087.
60. Widom, B., Potential-Distribution Theory and the Statistical-Mechanics of Fluids. *Journal of Physical Chemistry* **1982,** 86, (6), 869-872.
61. Lennard-Jones, J. E., Processes of adsorption and diffusion on solid surfaces. *Trans. Faraday Soc.* **1932,** 28, (333).
62. Jorgensen, W. L.; Chandrasekhar, J.; Madura, J. D.; Impey, R. W.; Klein, M. L., Comparison of Simple Potential Functions for Simulating Liquid Water. *Journal of Chemical Physics* **1983,** 79, (2), 926-935.
63. Jorgensen, W. L., Quantum and Statistical Mechanical Studies of Liquids .24. Revised Tips for Simulations of Liquid Water and Aqueous-Solutions. *Journal of Chemical Physics* **1982,** 77, (8), 4156-4163.
64. Jorgensen, W. L., Quantum and Statistical Mechanical Studies of Liquids .19. Pressure-Dependence of the Structure and Properties of Liquid Normal-Butane. *Journal of the American Chemical Society* **1981,** 103, (16), 4721-4726.
65. Berendsen, H. J. C.; Grigera, J. R.; Straatsma, T. P., The Missing Term in Effective Pair Potentials. *Journal of Physical Chemistry* **1987,** 91, (24), 6269-6271.
66. Glattli, A.; Daura, X.; van Gunsteren, W. F., Derivation of an improved simple point charge model for liquid water: SPC/A and SPC/L. *Journal of Chemical Physics* **2002,** 116, (22), 9811-9828.
67. Berendsen, H. J. C., Postma, J.P.M., van Gunsteren, W.F., Hermans, J., *Intermolecular Forces*. Reidel: Dordrecht, 1981.
68. *Materials Studio*, Accelrys, Inc. San Diego, CA USA.
69. Greenfield, M. L.; Theodorou, D. N., Geometric Analysis of Diffusion Pathways in Glassy and Melt Atactic Polypropylene. *Macromolecules* **1993,** 26, (20), 5461-5472.
70. Greenfield, M. L.; Theodorou, D. N., Coupling of penetrant and polymer motions during small-molecule diffusion in a glassy polymer. *Molecular Simulation* **1997,** 19, (5-6), 329-&.
71. Greenfield, M. L.; Theodorou, D. N., Molecular modeling of methane diffusion in glassy atactic polypropylene via multidimensional transition state theory. *Macromolecules* **1998,** 31, (20), 7068-7090.

72. Greenfield, M. L.; Theodorou, D. N., Coarse-grained molecular simulation of penetrant diffusion in a glassy polymer using reverse, and kinetic Monte Carlo. *Macromolecules* **2001,** 34, (24), 8541-8553.
73. Rabinovich, A. L.; Balabaev, N. K.; Alinchenko, M. G.; Voloshin, V. P.; Medvedev, N. N.; Jedlovszky, P., Computer simulation study of intermolecular voids in unsaturated phosphatidylcholine lipid bilayers. *Journal of Chemical Physics* **2005,** 122, (8).
74. Einstein, A., Investigations on the Theory of the Brownian Movement. *Annalen der Physik 17, page 549* **1905,** 17, 549.
75. Meier, K.; Laesecke, A.; Kabelac, S., Transport coefficients of the lennard-jones model fluid. I. Viscosity. *Journal of Chemical Physics* **2004,** 121, (8), 3671-3687.
76. Meier, K.; Laesecke, A.; Kabelac, S., Transport coefficients of the Lennard-Jones model fluid. III. Bulk viscosity. *Journal of Chemical Physics* **2005,** 122, (1).
77. Meier, K.; Laesecke, A.; Kabelac, S., A molecular dynamics simulation study of the self-diffusion coefficient and viscosity of the Lennard-Jones fluid. *International Journal of Thermophysics* **2001,** 22, (1), 161-173.
78. Metzler, R.; Klafter, J., The random walk's guide to anomalous diffusion: a fractional dynamics approach. *Physics Reports-Review Section of Physics Letters* **2000,** 339, (1), 1-77.
79. Sun, H., COMPASS: An ab Initio Force-Field Optimized for Condensed-Phase Applications Overview with Details on Alkane and Benzene Compounds. *J Phys Chem B* **1998,** 102, 7338-7364.
80. Jie Yang, Y. R., and An-min Tian, COMPASS Force Field for 14 Inorganic Molecules, He, Ne, Ar, Kr, Xe, H2, O2, N2, NO, CO, CO2, NO2, CS2, and SO2, in Liquid Phases. *J Phys Chem B* **2000,** 104, 4951-4957.
81. Stone, M. T.; da Rocha, S. R. P.; Rossky, P. J.; Johnston, K. P., Molecular differences between hydrocarbon and fluorocarbon Surfactants at the CO2/water interface. *Journal of Physical Chemistry B* **2003,** 107, (37), 10185-10192.
82. Song, W.; Rossky, P. J.; Maroncelli, M., Modeling alkane plus perfluoroalkane interactions using all-atom potentials: Failure of the usual combining rules. *Journal of Chemical Physics* **2003,** 119, (17), 9145-9162.
83. Duda, J. L.; Vrentas, J. S.; Ju, S. T.; Liu, H. T., Prediction of Diffusion-Coefficients for Polymer-Solvent Systems. *Aiche Journal* **1982,** 28, (2), 279-287.
84. Lee, K. M. Exploring Solvent Properties of High Pressure Carbon Dioxide via Computer Simulation. dissertation, University of Texas at Austin, Austin, 2003.

Vita

Frank Willmore was born March 4, 1971 in Melrose Park, Illinois, the youngest of six children of Floyd Earl Willmore and Josephine Dawn Willmore. He attended Oak Park-River Forest High School, earned a B.S. in chemistry at Valparaiso University and an M.S. in chemical engineering at Iowa State University before pursuing his doctorate at the University of Texas. He has worked as a software developer, designed and built a house, and regularly performs improv comedy.

Permanent address: c/o The Willmore Family, 216 S. Austin Blvd. Oak Park, IL 60304

This dissertation was typed by the author.